Extraktion von XML aus HTML-Seiten

Das WYSIWYG-Werkzeug d_2c

Diplomarbeit

Institut für
Programmstrukturen und Datenorganisation
Lehrstuhl Goos
Fakultät für Informatik
Universität Karlsruhe (TH)

Max Völkel

Betreuer:
Dipl.-Inform. Markus L. Noga

verantwortlicher Betreuer:
Prof. Dr. rer. nat. Gerhard Goos

Tag der Anmeldung: 24. Oktober 2002
Tag der Abgabe: 7. Mai 2003

Danksagung

Ich danke Markus L. Noga für seine gute Betreuung,
Rubino Geiß, Martina Haitz, Miriam Haitz, Bastian Hemminger und Frederik Thiele für ihre Hilfe beim Korrigieren.
Ganz besonders danke ich meiner Freundin Claudia, die mich auch in den arbeitsintensiven Phasen immer wieder zum Lachen gebracht hat.

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

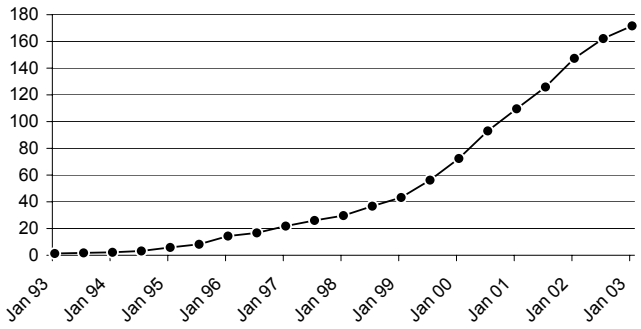
Karlsruhe, den 7. Mai 2003

Inhaltsverzeichnis

1 Einleitung

Wachstum von Informationsangebot und -nachfrage

Webseiten haben sich in den letzten Jahren als universale Schnittstelle zu Informationen wie Zeitungen und Nachrichten sowie interaktiven Auskunftsdiensten wie Suchmaschinen, Wettervorhersagen, Online-Auktionen, Telefonauskünften und Aktienkursen etabliert, und immer mehr Menschen nutzen diese Schnittstelle. Die Zahl der Menschen, die zu Hause das Internet benutzen, wuchs global von 563 Million im dritten Quartal 2002 auf 580 Millionen im vierten Quartal 2002 an [Nie02]. Gleichzeitig wächst auch die Menge der Rechner, die Informationen anbieten, weiter an (Siehe Abb. 1.1).



Angaben in Millionen.
DNS = *Domain Name System*
Quelle:
Internet Domain Survey,
Januar 2003.

Internet Software
Consortium
(http://www.isc.org/)

Abbildung 1.1: Im DNS eingetragene Rechner von '93 bis '03

Automatisierung notwendig

Diese große Informationsmenge kann nur durch Automatisierung genutzt werden:

> The Web can reach its full potential only if it becomes a place where data can be shared and processed by automated tools as well as by people. [sww01]

Um die Inhalte von Webseiten für andere Systeme nutzbar zu machen, muss die Semantik der Inhalte explizit gemacht werden, so wie z. B. XML[1] dies durch eine Baumstruktur tut. Ein Beispiel dafür sind automatische Preisvergleicher, die aus hunderten von Händler-Seiten die Preisangaben extrahieren und das günstigste Angebot ausgeben.

Anbieter haben kein Interesse an Datenextraktion

Auch das Konzept des *Semantic Web* sieht eine *anbieter*seitige Annotation der Daten vor. Die Anbieter haben aber teilweise kein Interesse daran und werden – ganz im Gegenteil – eher versuchen eine automatische Datenextraktion zu verhindern:

[1] Extensible Markup Language

> It is not always in the interest of web site owners if their data is extracted and used for third party purposes, especially taking into account the widespread commercial use of advertising banners that would be bypassed by these extraction procedures. It is, therefore, questionable whether all HTML pages will be replaced by XML documents in the near future. [KT03]

Kosten

Für manche, besonders gewachsene Systeme, ist die nachträgliche anbieterseitige Aufbereitung der Inhalte auch mit hohen Kosten verbunden.

*Anwender*seitige Aufbereitung und Extraktion

Daher muss die Aufbereitung und Extraktion von Daten, die in HTML[2] vorliegen, *anwender*seitig erfolgen. Ein Programm zur Aufbereitung und Extraktion aus Quelldaten wird in diesem Zusammenhang als *Wrapper* bezeichnet. Zur einfachen Integration in bestehende Systeme sollte ein Wrapper sein Extraktionsergebnis online als XML über einen Webserver zur Verfügung stellen.

Erstellen von Wrappern

Die Erstellung von Wrappern ist mühsam:

> Navigation rules and extraction rules are currently optimized by hand, a burden that automatic or semiautomatic tools may ease. [Myl01]

WYSIWYG: einfacher, weniger Fehler

Ein Wrapper soll möglichst einfach zu erstellen, zu nutzen und zu reparieren sein. Das kann nur über einen WYSIWYG-Ansatz geschehen, bei dem die Wrapper mit dem Browser aus einer Darstellung der Quellseite (zu extrahierende Seite) interaktiv erzeugt werden, in einer Weise, die dem normalen Navigationsverhalten des Benutzers auf HTML-Seiten entspricht. Da keine Kontextwechsel zwischen verschiedenen Anwendungen stattfinden, kann sich der Benutzer stärker auf die Auswahl von zu extrahierenden Elementen konzentrieren. Zudem ist ein geeigneter WYSIWYG-Ansatz weniger fehleranfällig, da die Syntax der Extraktionsanweisungen und des HTML-Quelltextes vor dem Benutzer verborgen wird. Dadurch kann statt einer Reparatur bequem ein neuer Wrapper erstellt werden, wodurch der Benutzer kein weiteres Verfahren erlernen muss.

Anmerkung:
WYSIWYG steht für „What You See Is What You Get“ und bezeichnet die Idee, auf der Darstellung der Daten zu arbeiten. Der Benutzer sieht dabei während der Bearbeitung eine Darstellung, die dem Endergebnis möglichst ähnlich sieht. Ursprünglich wurde dieser Begriff für Textverarbeitungen und Layoutanwendungen geprägt.

[2] Hypertext Markup Language

1.1 Aufgabenstellung

Die Aufgabe besteht darin, ein System zu erstellen, mit dem *Wrapper* zur Datenextraktion aus Webseiten möglichst einfach erzeugt werden können. Dabei sollen die von marktbeherrschenden Browsern erzeugten visuellen Darstellungen der Webseiten in zweierlei Weise als Vorbild dienen. Zum einen sollen möglichst alle dargestellten HTML-Elemente als entsprechendes strukturiertes, wohlgeformtes XML extrahiert werden können. Zum anderen soll die Erstellung von Wrappern sich möglichst wenig von der gewohnten Naivgation im Internet unterscheiden. Das Extraktionsergebnis soll zur einfachen Integration in bestehende Systeme als XML über eine Webschnittstelle abrufbar sein. Das Gesamtsystem soll eine modulare Architektur aufweisen und die plattformunabhängige Ausführung von Wrappern ermöglichen. Das erstelle System wird im folgenden mit d_2c bezeichnet[3].

[3] d_2c (document to content) – Ein WYSIWYG-Werkzeug zur Erstellung und Nutzung von Wrappern auf Webseiten

2 Grundlagen

In diesem Kapitel wird die Bedeutung der Browser aus Sicht der Benutzer betrachtet. Sie dienen als Vorbild für die Benutzer-Schnittstelle von d_2c und den Umgang mit real vorkommendem HTML. Dieses wird analysiert und als häufig fehlerhaft erkannt. Die Interpretation von HTML wird als eine nicht-triviale Zerteileraufgabe erkannt. Bei der Betrachtung von Webseiten aus Datenbanken werden die Begriffe *Struktur* und *Inhalte* definiert. Wrapper (Definition in 2.4) nutzen die *Struktur* einer Webseite aus, um die *Inhalte* zu extrahieren. Die einzelnen Teilaufgaben und mögliche Probleme in der Praxis werden beleuchtet, um die wichtigsten Probleme zu identifizieren. Abschließend werden Kriterien für ein Werkzeug zur Erstellung von Wrappern angegeben und die Aufgabenstellung verfeinert.

2.1 Die Rolle der Browser

Browser zur visuellen Datenintegration

Das Internet – genauer: das *World Wide Web* (WWW) – ist nur mit Hilfe eines Browsers nutzbar. Diese sind heute sehr mächtige Werkzeuge. Sie integrieren visuell die unterschiedlichsten Datenformate und Programmiersprachen:

- HTML wird interpretiert und grafisch dargestellt.
- CSS[1] ist eine Sprache, um die visuelle Darstellung von HTML im Detail zu beschreiben. Die zu formatierenden Elemente werden mit einer eigenen Beschreibungssprache ausgewählt. Das W3C[2] hat CSS spezifiziert, um die HTML-Daten und ihre Darstellungsparameter stärker voneinander zu trennen.
- JavaScript kann interaktiv eine bestehende HTML-Seite nach belieben inhaltlich und visuell ändern.
- Java-Applets ermöglichen das Abarbeiten von Java-Programmen innerhalb eines definierten rechteckigen Bereichs im Browser-Fenster.
- *Flash* und *Shockwave* sind interaktive Vektorgrafiken, die eine proprietäre Skriptsprache benutzen. Sie können zusätzlich oder alternativ zu HTML verwendet werden.
- Weitere *Plugins* können beliebige Inhalte visuell darstellen.

[1] Cascading Style Sheets
[2] World Wide Web Consortium, `http://www.w3.org/`

Klick-Konzept als kleinster gemeinsamer Nenner

Den kleinsten gemeinsamen Nenner der verschieden gestalteten Webseiten bildet dabei das Klick-Konzept: Ein interessantes Element – sei es Navigationsknopf, Verweis oder Bild – wird angeklickt und meist passiert daraufhin etwas. Oft wird eine neue Seite geladen, die genauer auf das ausgewählte Element eingeht.

Fazit: Ein Browser ist für den Benutzer das Standardwerkzeug zur Navigation im Internet. Er ist ein Werkzeug zur visuellen Datenintegration mit dem zentralen Konzept „Anklicken bei Interesse".

Im nächsten Abschnitt wird beleuchtet, was ein Browser tut, um dem Benutzer eine visuelle Darstellung der Daten zu ermöglichen.

2.2 HTTP und HTML in der Praxis

Die beiden Standards HTTP[3] und HTML werden in der Praxis kaum korrekt verwendet, sondern meist mit verschiedenen, nicht-standardisierten und undokumentierten Erweiterungen angereichert.

HTTP

In einer Stichprobe von 9000 zufälligen URLs[4] kommen bereits über 50 verschiedene HTTP-Antwort-Header vor – HTTP 1.1 spezifiziert aber nur 28.

HTML in der Praxis selten korrekt

Wie standardkonform sind Webseiten in der Praxis? Um dieser Frage nachzugehen wurden einige tausend Webseiten heruntergeladen und auf gültige HTML-Syntax überprüft (Genaues Vorgehen wird in 5.3 beschrieben). Korrekt im Sinne der HTML 4.0-Spezifikation des Gremiums W3C sind nur 0,24 % der Webseiten. Der überwiegende Teil der Daten im WWW, welche mit einem Browser betrachtet werden können, liegt in einem HTML vor, welches nach dem HTML 4.0-Standard ungültig ist.

Definition von PH

Reales HTML ist syntaktisch de facto in den meisten Fällen eine Zeichenkette und keine Baumbeschreibung. Um diesen Unterschied deutlich zu machen, wird in der Praxis auf Webseiten vorkommendes HTML in dieser Arbeit mit PH bezeichnet. Das „P" steht für „Pseudo" oder „Praxis", „H" für HTML. PH ist formal eine Zeichenkette, *intentional* ist es HTML.

Browser zerteilen PH unterschiedlich

Ein Browser interpretiert beim Zerteilen die PH-Zeichenkette und erzeugt daraus eine interne Baumstruktur, welche dann visuell dargestellt wird. Verschiedene Browser interpretieren PH unterschiedlich. Das bedeutet, dass jeder Browser intern einen anderen HTML-Syntaxbaum erzeugt.

Der Chef und Gründer von *Opera*, dem drittgrößten Browser-Hersteller nach Marktanteilen, sieht dies als Problem für seine Branche und erklärt, wie es entstanden ist:

> ... weil sich 95 Prozent der Sites nicht an die Standards halten. *Mosaic* versuchte als Erster, nett zu Web-Autoren zu sein und deshalb

[3] Hypertext Transfer Protocol
[4] Uniform Ressource Locator

> zu erraten, was der Seitenersteller wirklich wollte. *Netscape* und *Microsoft* machten genauso weiter. Mit diesem Problem wird sich jeder, der in den Browser-Markt will, sehr schwer tun. [Sch]

PH-Struktur zur Extraktion

Nur ein Mensch kann einer vom Browser grafisch aufbereiteten Darstellung von PH die Pragmatik entnehmen. Da eine automatische Analyse der pragmatischen Ebene technisch voraussichtlich nie möglich sein wird, können nur die vorhandenen Informationen in der PH-Struktur, zur automatischen Verarbeitung der Inhalte genutzt werden.

PH muss wie von einem Browser interpretiert werden

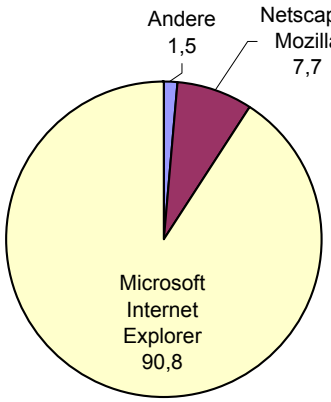


Abbildung 2.1: Marktanteile von Browsern in Deutschland
Angaben in Prozent. Quelle: fittkaumaas.de, 15. WWW-Benutzer-Analyse W3B, 2003

Da kein Browser alle Spezifikationen für HTML, JavaScript und CSS korrekt und vollständig erfüllt[5], können Seitenautoren nur durch einen visuellen Test mit einen Browser überprüfen, wie ihre Seite dargestellt wird. Die „Spezifikation“ von PH ist daher nur durch die Implementierungen in marktbeherrschenden Browsern gegeben (siehe Abb. 2.1) und diese müssen auch als Definition einer korrekten Zerteilung von PH herangezogen werden. Verbreitete Browser sind *Internet Explorer* (Microsoft), *Netscape* (AOL), *Mozilla* (Open Source) und *Opera* (Opera).

2.3 Struktur und Inhalte von Webseiten

Herkunft von PH

Nach [SA99] werden 80 % der Webseiten aus Datenbanken erstellt. Die übrigen 20 % der PH-Dokumente sind teilweise handgeschrieben, größtenteils jedoch mit spezialisierten HTML-Editoren erstellt worden. Diese Seiten sind statisch und für eine automatisch Datenextraktion nicht interessant, da die gewünschten Daten am schnellsten manuell im Browser markiert und kopiert werden können.

Webseiten aus Generatoren ähneln sich vom Aufbau her stark und besitzen oft gemeinsame Elemente, z. B. Navigationsleisten, den Seitentitel oder Copyright-Hinweise. Auf einer Webseite kommen dabei entweder mehrere ähnliche Seiten vor oder eine Seite ändert sich zu bestimmten Zeitpunkten. In beiden Fällen werden die Seiten mit Hilfe von Schablonensystemen erstellt. Da die Schablonen meist per Hand erstellt werden, sind auch die von ihnen erzeugten Seiten selten korrektes HTML. Interessant für die Extraktion sind die veränderlichen Inhalte, die innerhalb der Schablonenstruktur liegen. Wrapper nutzen diese Struktur aus,

[5] `http://www.webstandards.org`

um die Inhalte zu erkennen. Demzufolge kann ein Wrapper, der für eine solche Seite erstellt wurde, auch auf den anderen Seiten genutzt werden, die auf gleiche Weise aus der Datenbank erzeugt wurden.

Definition von *Struktur* und *Inhalten*

Gegeben sei eine Menge M von Webseiten, die aus einem speziellen datenerzeugenden Prozess stammen. Dieser Prozess sei ein Webserver, der Informationen aus einer externen Quelle, wie einer Datenbank, über Schablonen in HTML-Seiten verpackt. Dabei wird die Annahme gemacht, dass die erzeugten Webseiten ein einheitliches Erscheinungsbild aufweisen, um für den Benutzer der Webseiten leicht bedienbar zu sein. Des weiteren wird davon ausgegangen, dass die Webseiten mit möglichst geringem Aufwand erzeugt werden und daher möglichst einfach aufgebaut sind. Nun werden folgende Begriffe definiert:

Gleichartige Seiten sind alle Seiten der Menge M. Daher weisen alle gleichartigen Seiten eine gemeinsame Struktur auf, die sich z. B. durch eine DTD[6] oder ein XML-Schema beschreiben lässt. Diese Strukturbeschreibung muss einem Wrapper zur Datenextraktion nicht bekannt sein.

Die Struktur einer Webseite ist definiert als der mit allen Seiten aus der Menge M gemeinsame Schnittbaum der HTML-Elemente, auch reine Text-Elemente gehören dazu.

Die Inhalte einer Seite sind die unterhalb des Strukturbaumes vorkommenden Unterbäume.

Die Grafik 2.2 verdeutlicht die Definition: Zwei Webseiten A und B werden als HTML-Elementbäume dargestellt, der dritte Baum zeigt die gemeinsame *Struktur* (grau) und die darunter liegenden *Inhalte* (schwarz).

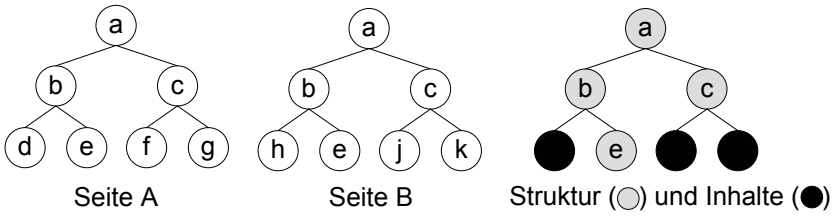


Abbildung 2.2: Struktur und Inhalte von Webseiten

2.4 Wrapper

Definition eines Wrappers

Was ist ein Wrapper? Die Definitionen aus der Literatur bieten kein einheitliches Bild:

- "[...] *extracts information* from the Web pages written in a specific format" [Coh99],

[6] Document Type Definition

- "[...] *transform data* from less structured representation into a more structured representation" [LPH00],
- "[...] offer *high-level view and access* to some data. Using them [the wrappers], the data can be handled transparently in a uniform and structured way." [SA01],
- "[...] automatically *extract data* from Internet websites and convert the information into a structured format [KT03]".

Definition der Softwaretechnik

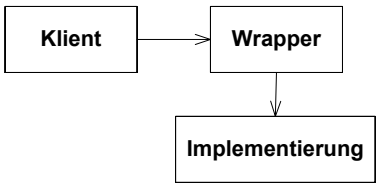


Abbildung 2.3: Das Entwurfsmuster *Wrapper* in der Softwaretechnik

Die Softwaretechnik definiert mit dem Entwurfsmuster *Wrapper* [GHJV94] eine Zwischenschicht zwischen einem bestehendem System und einem Klient (siehe Abb. 2.3). Sie haben die Aufgabe, zwei Schnittstellen zueinander kompatibel zu machen. Im Unterschied zum ähnlichen Entwurfsmuster *Adapter* ist beim *Wrapper* nicht im voraus bekannt, wer das aufrufende Objekt sein wird. Der *Wrapper* konzentriert sich darauf, von der Schnittstelle der Implementierung zu abstrahieren und eine neue, höhere Schnittstelle anzubieten.

Verwendete Definition eines Wrappers im Internet

Im Kontext des Internets werden mit Wrappern Software-Artefakte bezeichnet, die von den Basistechnologien HTTP und HTML abstrahieren und als Schnittstelle Dokumente oder ausgewählte Teile von Dokumenten zurückliefern. Dies ist in dieser Arbeit mit dem Begriff *Wrapper* gemeint. Derart definierte Wrapper sollen in bestehende Systeme integriert werden. Welche Datenformate und Schnittstellen sind dazu geeignet?

XML und Webdienste als universale Schnittstelle

XML hat als Datenaustauschformat zur Kommunikation zwischen verschiedenen Systemen und Plattformen eine sehr große Bedeutung erlangt. Es eignet sich besonders für den Austausch von als Bäumen strukturierten Daten. Für nahezu jede Programmiersprache sind Programmier-Bibliotheken zur Bearbeitung von XML verfügbar. Die neuen, auf XML aufbauenden Webdienste (im engl: *web service*), sind als Integrationstechnologie für die lose Kopplung von Softwaresystemen konzipiert, werden aber in der Praxis bisher kaum eingesetzt, da es sich noch um eine sehr junge Technologie handelt. Ein Wrapper wird stets in ein umgebendes System eingebunden und sollte sich deshalb besonders leicht integrieren lassen. Daher sollte er die von ihm extrahierten Daten als XML-Dokument über den Aufruf einer Webadresse zurückliefern können. Dabei wird auf den Einsatz komplexer Protokolle wie SOAP[7] zugunsten einfacherer Integration verzichtet. Ein Wrapper, der über so eine einfache Webschnittstelle genutzt werden kann, wird in dieser Arbeit auch als *Dienst* bezeichnet.

[7] Simple Object Access Protocol

2.4.1 Atomare und höhere Dienste

Bei der Nutzung von Wrappern als Dienste können zwei Ebenen, *atomare* und *höhere* Dienste, unterschieden werden [LPH00].

Atomare Dienste entsprechen der Abstraktion von einer einzelnen Webseite. Die Quellseite, aus der hierbei Daten extrahiert werden sollen, ist über eine einzige HTTP-Anfrage erreichbar.

Zu den Aufgaben atomarer Dienste zählen die transparente Kommunikation mit den Webservern, die Aufbereitung des erhaltenen PH-Textes in eine Baumstruktur und die Datenextraktion aus dieser. Das Ergebnis eines *atomaren Dienstes* ist ein über eine Webschnittstelle abrufbares XML-Dokument, welches die extrahierten Daten der Quellseite enthält. Diese Schnittstelle kann auch über einen Webdienst ermöglicht werden.

Höhere Dienste werden aus atomaren oder anderen höheren Diensten komponiert. Navigation zwischen Seiten und Integration von Daten verschiedener Seiten sind Aufgaben der höheren (zusammengesetzten) Dienste.

Anmerkung:
Das Zusammenfassen der Ergebnisse verschiedener atomarer Dienste kann durch ein XSLT[8]-*Stylesheet*[9] realisiert werden. Zur Navigation rufen die *höheren Dienste* mehrere atomare Dienste auf.

Durch die Kopplung über eine Webschnittstelle können die Aufrufe in einer beliebigen Programmiersprache realisiert werden. Dabei sind das *Session Management*[10] und *Cookies* zu berücksichtigen. Es stellt sich die Frage, ob solche höherwertigen Dienste automatisch erstellt werden können, wenn der Benutzer bei der Interaktion mit den atomaren Diensten maschinell beobachtet wird (siehe 8.3).

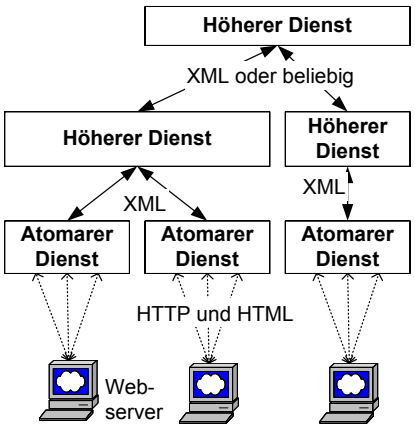


Abbildung 2.4: Zusammenspiel der *atomaren* und *höheren* Dienste

Fazit: Die *atomaren Dienste* abstrahieren vollständig von der zugrunde liegenden Datenstruktur und die *höheren Dienste* nutzen diese Abstraktion (vergl. Abb. 2.4).

[8] eXtensible Stylesheet Language Transformations
[9] Ausgabeformatschablone
[10] Persistente Verbindung zwischen Browser und Webserver auf einer höheren logischen Schicht, da HTTP selbst solche Mechanismen nicht bietet.

2.4.2 Beispiele typischer Aufgaben

Drei verschiedene Typen von Webseiten und dazugehörigen Aufgaben können bei Wrappern unterschieden werden:

Text

Vom Aufgabentyp „Text“ sind Artikel und andere Texte, wie sie typischerweise auf den Seiten von Zeitungen und anderen Online-Medien vorkommen. Die Struktur eines solchen Textabschnittes spielt für die Extraktion keine Rolle, sollte aber im Extraktionsergebnis erhalten bleiben. Ein Benutzer interessiert sich bei der Extraktion des Leitartikels, aus dem Online-Angebot einer Tageszeitung, meist für den vollständigen Artikel inklusive seiner Formatierungen. Einige Beispiele für Text-Aufgaben finden sich in Tabelle 2.1.

Seite	*Extraktionsaufgabe*
Nachrichtenticker www.heise.de/newsticker	Alle Nachrichten ohne Navigation und Werbebanner extrahieren, z. B. zur Einbindung in eine andere Webseite oder Filtern nach Stichwörtern
amerikanischer Fernsehsender www.cnn.com	Extraktion der neusten Meldungen („Top Stories“)
Schneehöhen-Bericht www.lifte.tv/anzeige_ort.php	Extraktion der Schneehöhe und Temperatur auf dem Feldberg, z. B. zur Benachrichtigung sobald genug Schnee zum Skifahren vorhanden ist oder zur Einbindung in eine Schneehöhen-Suchmaschine
Vorlesungs-Homepage eins.info.uni-karlsruhe.de	Extraktion der neusten Meldung aus dem Bereich „Aktuelles“, z. B. zur automatischen Benachrichtigung per Email

Tabelle 2.1: Beispiele für Aufgaben vom Typ Text

Datensatz

Der Typ „Datensatz“ bezeichnet die Extraktion aus Webseiten, die nach einer (Such-)Anfrage des Benutzers, mit Hilfe von Informationen aus einer Datenbank, dynamisch erzeugt werden. Wenn der Benutzer verschiedene Anfragen mit dem selben Wrapper durchführen möchte (z. B. Telefonauskunft für verschiedene Nachnamen), dann muss dieser zur Laufzeit eine parametrisierbare HTTP-Anfrage an den Quelldienst senden können. Beispiele für Datensatz-Aufgaben finden sich in Tabelle 2.2.

Fernsteuerung

Mit „Fernsteuerung“ werden Aufgaben auf stark interaktiven Seiten bezeichnet, z. B. bei freien Email-Anbietern, Online-Auktionshäusern oder der Fernsteuerung einer realen Maschine. Hier ist zusätzlich zur Datenextraktion noch eine automatisierte Bedienung gefragt, z. B. das Verfolgen von Links oder Absenden von Formularen. Weitere Beispiele finden sich in Tabelle 2.3. Die Aufgaben reichen von reinen Extraktionsaufgaben (Typ Text) über einschrittige Interaktion (Typ Datensatz) bis zu komplexen Navigationsaufgaben (Typ Fernsteuerung).

In der Praxis vermischen sich diese drei Aufgabentypen teilweise. Jemand, der beispielsweise eine Übersicht über seine verschiedenen Online-Konten erstellen möchte, muss zunächst eine Anmeldung auf den Online-Angeboten der einzelnen Banken simulieren (Fernsteuerung) und danach die Kontostände extrahieren

Seite	*Extraktionsaufgabe*
Telefonbuch www.telefonbuch.de	Extraktion der Suchergebnisse zu einem dynamisch angegebenen Nachname und Wohnort
Aktienkurse www.quote.com	Extraktion des Aktienkurses für ein gegebenes Aktienkürzel
Buchrecherche www.amazon.de	Preis und Titel des ersten Buchs für die Suchanfrage „webservices“ z. B. für eine Preisbeobachtung
CD-Datenbank www.gracenote.com/music	Zu einem gegebenen Titel soll der Künstler gefunden werden, der den Titel erstellt hat
Google www.google.de	Extraktion der Anzahl gefundener Treffer, z. B. zur Bestimmung der korrekten Schreibweise eines Wortes (häufigere Wörter sind wahrscheinlich richtig)
Wettervorhersage www.wetteronline.de	Für eine gegebene Postleitzahl die Höchst- und Tiefsttemperaturen des nächsten Tages bestimmen
MP3-Datenbank www.audiogalaxy.com	Suchen nach zu einem Künstler ähnlichen Künstlern („other listeners liked“)
Auktionen www.ebay.de	Extraktion von Espressomaschinen-Preisen
Zufällige URL www.mangle.ca	Extraktion genau einer zufälligen URL, z. B. zum Erzeugen von Testdaten

Tabelle 2.2: Beispiele für Aufgaben vom Typ Datensatz

Seite	*Extraktionsaufgabe*
Handelsspiel www.uga-agga.de	anmelden, Spielhandlung ausführen, Punktestand extrahieren
Jobbörse mein.monster.de	anmelden, neues Jobangebot extrahieren
Webmail-Service www.gmx.de	anmelden, Email versenden
Uni-Bibliothek Karlsruhe www.ubka.uni-karlsruhe.de	Anmelden und eine Pauschalverlängerung durchführen

Tabelle 2.3: Beispiele für Aufgaben vom Typ Fernsteuerung

(Datensatz).

Beispiele im Kontext der *atomaren* und *höheren* Dienste

Von den hier definierten Aufgaben sind die reinen Extraktionsaufgaben (Typen Text und Datensatz) den *atomaren Diensten* zuzuordnen. Wenn ein System aus einer Seite beliebigen Inhalt korrekt extrahieren kann, dann kann ein darauf aufbauendes die enthaltenen Verweise und Formulare zur Navigation nutzen (Typ Fernsteuerung). Daher beschränkt sich diese Arbeit auf die Extraktion aus Webseiten, die direkt über einen Verweis oder das Abschicken eines Formulars erreicht werden können.

2.4.3 Modell eines atomaren Dienstes

Die Aufgabe eines Wrappers lässt sich nach [SA99] in drei Teilaufgaben gliedern:

1. Herunterladen der HTML-Seite aus dem Online-Angebot
2. Identifikation und Extraktion von Informationen, die in den HTML-Seiten nur implizit dargestellt sind. In dieser Arbeit wird ein Schwerpunkt auf diesen Schritt gelegt.
3. Speichern der extrahierten Daten zur Weiterverarbeitung, z. B. in XML oder in einer Datenbank.

Die zweite, komplexe Aufgabe der Datenextraktion kann in zwei Teilaufgaben weiter zerlegt werden:

- Zunächst ist PH mit Hilfe eines fehlertoleranten Zerteilers zu einer Baumstruktur aufzubereiten und
- *anschließend* sind aus der Baumstruktur Daten zu extrahieren.

Zahlreiche Wrapper-Werkzeuge versuchen beide Schritte auf einmal zu machen und aus der PH-Zeichenkette direkt Daten zu extrahieren. Dabei benutzen sie reguläre Ausdrücke, Perl-Programme oder eigene Zeichenketten-Beschreibungssprachen. Die erzeugten Wrapper sind damit monolithische Systeme, zwischen denen eine Wiederverwendung von Teilkomponenten nicht ohne weiteres möglich ist. Ein Vergleich zwischen verwendeten Extraktionssprachen findet sich im Abschnitt 3.1.

2.4.4 Wrapper in der Praxis

In der Praxis verändern sich Webseiten nicht nur *inhaltlich* sondern auch *strukturell* oder verschwinden ganz. Die häufigste Veränderung einer Webseite ist die Umordnung ihrer Elemente [Myl01]. Da Wrapper die Struktur zur Datenextraktion nutzen, ist es wichtig zu untersuchen, wie oft die Änderungen stattfinden. Über die Art und Häufigkeit von *strukturellen* Änderungen liegen nur wenig Daten vor:

Änderungen finden selten statt

Bei einer Untersuchung [Kus99] wurden für 27 Seiten sehr einfache Wrapper (basierend auf einfachen regulären Ausdrücken) erstellt und über 6 Monate beobachtet. Bei 56 % der getesteten Seiten änderte sich die Seite *strukturell* gar nicht. Auf den anderen Seiten fanden im Durchschnitt 1,9 *strukturelle* Änderungen (maximal 4) statt. Die *Struktur* einer beliebigen Webseite ändert sich damit hochgerechnet im Mittel nur zweimal im Jahr. Mittlerweile dürfte diese Frequenz sogar noch gesunken sein, wie *Schneider*[11] ausführt:

[11] Dipl.-Inform. P. Schneider, persönliche Mitteilung, 4.2.2003, Karlsruhe. P. Schneider ist Mitarbeiter von *Rapid Solution*, eine Firma die seit 1999 u.A. Meta-Suchmaschinen (*Equero*) entwickelt und betreibt.

> Webseiten ändern sich jetzt nicht mehr so stark, wie zu den Zeiten der *New Economy* in den Jahren 2000/2001.

Eine höhere Änderungsrate berichtet allerdings [Kus00]:

> Excite's „Jango“ shopping agent, for example, relied on several hundred wrappers, each with a mean time to failure of about one month

Fazit: Die für Wrapper relevante Struktur einer Webseite verändert sich zwei bis zwölf Mal im Jahr. Da die Änderungsrate insgesamt abnimmt, hat dieses Problem eine geringer werdende Bedeutung.

Robuste Wrapper

Unter einem *robusten Wrapper* versteht man einen Wrapper, der auch bei Veränderungen der Quellseite möglichst die korrekten Daten extrahiert. Je „robuster“ ein Wrapper ist, um so besser kann er Änderungen an der Quellseite adaptieren. Aber welche Daten liefern sie dann? Es ist – unabhängig von der verwendeten Extraktionssprache – nicht möglich, mit Sicherheit die richtigen Daten zu extrahieren, wenn die Seite sich strukturell ändert, da der Wrapper die Pragmatik der Extraktionsaufgabe nicht verstehen kann. Eine Adaption auf veränderte Quellseiten kann nur über potentiell fehleranfällige Heuristiken erreicht werden.

Anbieter gegen Wrapper

Manche der *strukturellen* Änderungen auf Webseiten sind nicht die Folge regulärer Aktualisierungen sondern bewusster Maßnahmen gegen eine Datenextraktion durch Wrapper. Prinzipiell können sich Webautoren zu einem gewissen Grad dagegen wehren, dass ihre Inhalte von Wrappern gelesen werden können, indem die Seiten nicht in PH sondern z. B. in *Flash* gestaltet werden. Damit werden allerdings auch Benutzer, die keinen *Flash*-fähigen Browser benutzen, ausgeschlossen. Eine Webseite, die kompatibel für eine breite Zielgruppe ist, ist auch kompatibel für einen Wrapper.

2.5 Zusammenfassung

Browser sind das Standardwerkzeug zum Betrachten von HTML und zur Navigation im Internet. In der Praxis wird HTML jedoch fast nie standardkonform verwendet. Daher sollte ein Wrapper PH mit Hilfe eines fehlertoleranten Zerteilers aufbereiten, um die hierarchische Struktur von HTML zu erhalten. Diese Struktur wird von Wrappern genutzt, um die darin liegenden Inhalte zu extrahieren. Die Teilaufgaben eines Wrappers können in zwei Ebenen (*atomare* und *höhere*) zerlegt werden. Ein *atomarer Dienst* lässt sich weiter in die Teilaufgaben zerlegen.In der Praxis ändern sich Webseiten heute seltener als monatlich. *Robuste Wrapper* adaptieren zu einem gewissen Grad auf die Veränderungen an den Quellseiten.

Zum Ende der Einleitung werden Kriterien für ein Werkzeug zur Wrapper-Erstellung genannt, welche die Aufgabenstellung verfeinern.

2.6 Kriterien

K 1 *Eigenschaften erzeugter Wrapper*

K 1a Die erzeugten Wrapper sollen die Klasse der *atomaren Dienste* beherrschen. Möglichst jede von einem Browser dargestellte Seite soll als Quellseite zur Extraktion der *Inhalte* verwendbar sein.

K 1b Die erzeugten Wrapper sollen PH möglichst wie ein Browser interpretieren und in eine Baumstruktur überführen, aus der die Daten schließlich extrahiert werden.

K 2 *Erstellung von Wrappern*

Die Wrapper-Erstellung soll möglichst einfach durchführbar sein. Dazu soll eine WYSIWYG-Schnittstelle im Browser verwendet werden, die der gewohnten Navigation im Internet möglichst ähnlich ist. Kontextwechsel zwischen Browser und anderen Oberflächen sind so weit wie möglich zu vermeiden, damit der Benutzer sich stärker auf die Wrapper-Erstellung konzentrieren kann.

K 3 *Kompatibilität und Wiederverwendung*

K 3a Das Ergebnis eines Wrapper soll als XML-Dokument über eine Webschnittstelle nutzbar sein, um möglichst einfach in bestehende Systeme integrierbar zu sein.

K 3b Von Fortschritten in einzelnen Teildisziplinen der Wrapper-Erstellung soll das Gesamtsystem zeitnah profitieren können. Einzelne Komponenten sollen zu bestehenden Standards kompatibel sein, damit der Benutzer auch bestehende Werkzeuge benutzen kann.

3 Stand der Technik

In diesem Kapitel werden zunächst bestehende Systeme zur Erzeugung von Wrappern unter dem Gesichtspunkt der verwendeten Extraktionssprachen verglichen. In der zweiten Hälfte werden PH-Zerteiler vorgestellt.

3.1 Systeme zur Wrapper-Erzeugung

Eine gute Übersicht über bestehende Wrapper-Systeme von 1997 bis heute liefern [LRNST02, KT03]. Die folgenden Angaben sind im wesentlichen dort entnommen. Von den aufgelisteten 32 Wrapper-Systemen können 21 (davon 7 nichtkommerziell) das wichtige Ausgabeformat XML (Kriterium K 3a) liefern. Dem Überblick über die Entwicklungen in diesem Umfeld (siehe Abb. 3.1) kann entnommen werden, dass es 1998 ein großes Interesse an Wrapper-erzeugenden Systemen gab, aber keiner der frühen Ansätze weiterentwickelt wurde.

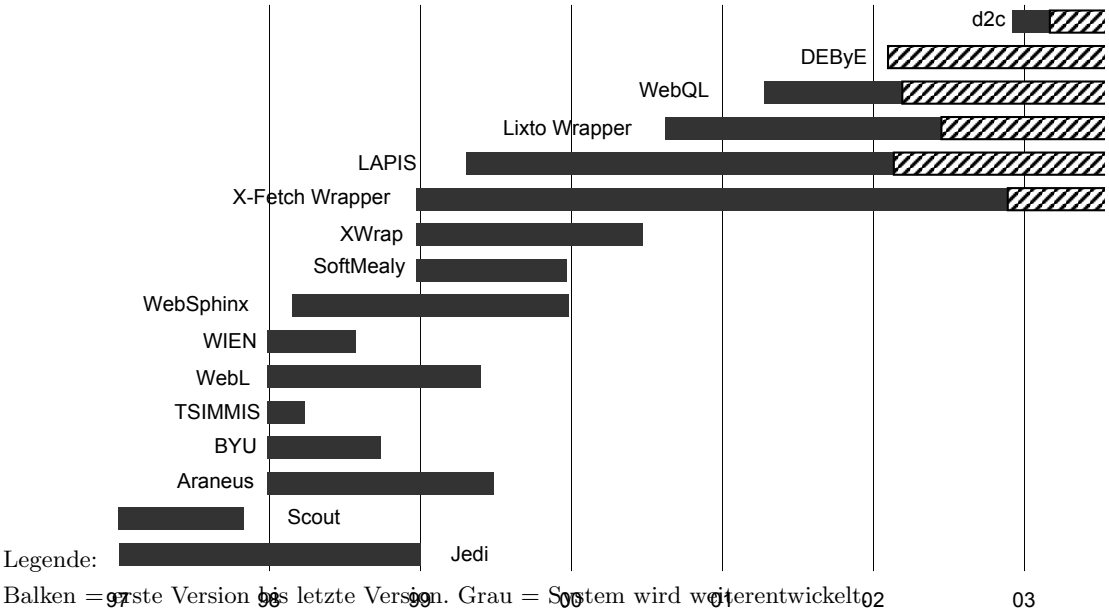


Abbildung 3.1: Vergleich ausgewählter Wrapper-Systeme

Folgende Systeme werden weiterentwickelt: der *X-Fetch Wrapper* vom Anbie-

ter Republica, LAPIS von der Carnegie Mellon University, das *Lixto Wrapper Generation Toolkit* von der gleichnamigen Firma Lixto (im Folgenden kurz *Lixto* genannt), *WebQL* [LL01] von QL2 Software[1] und *DEByE* von der Università di Roma Tre. Zusätzlich sollen noch das kommerzielle *ParserStudio* vom Anbieter *ItemField* (für das keine Versionsangaben vorliegen) und als „Klassiker" das nicht-kommerzielle System *XWrap* von der University of Maryland untersucht werden. Alle diese Systeme können XML ausgeben.

DEByE[2] (Date Extraction By Example, befindet sich noch in der Entwicklung) [GLdSRN00] generiert Wrapper aus Beispielvorgaben des Benutzers [LRNdSS00]. Zur Extraktion werden automatisch generierte reguläre Ausdrücke auf dem PH-Quelltext benutzt und kein spezieller PH-Zerteiler. Aus diesem Grund ist auch keine Modularisierung im Sinne des Kriteriums K 3b realisiert. Als Benutzerschnittstelle werden instrumentalisierte PH-Seiten und eine Java-Anwendung verwendet, in welcher der Benutzer in einem Graph von Webseiten navigieren kann. Die Ausgabe ist alternativ zu XML auch in eine SQL-Datenbank möglich.

LAPIS[3] ist ein Werkzeug zum Editieren von Text [MM02], kann aber auch zur Wrapper-Erzeugung genutzt werden. Es besitzt einen integrierten, programmierbaren Browser (augenscheinlich eine Eigenentwicklung in Java) mit einer Kommandozeile, die *Tcl*-Befehle und -Skripte verarbeitet. Darüber kann eine Einbettung von anderen Komponenten erfolgen. Als Extraktionssprache wird die eigene Sprache „Text Constraints" benutzt, die genauso ausdrucksstark wie reguläre Ausdrücke ist, deren Syntax jedoch „extrem intuitiv und einfach zu verstehen" [KT03] ist. Die konzeptuelle Grundeinheit der Sprache ist ein Text-Intervall, welches durch Zerteiler, reguläre Ausdrücke oder Grammatiken definiert wird. *LAPIS* verwendet zum Zerteilen von PH eine eigene Implementierung (1282 echte Quelltextzeilen), die vermutlich nicht an die Qualität spezialisierter, weitaus umfangreicherer Implementierungen heranreicht (siehe nächsten Abschnitt und Seite 38).

```
first Line in Java.Comment just before Java.Method
containing „toString"
```

Abbildung 3.2: Beispiel für einen „Text Constraints"-Ausdruck in *LAPIS*

Lixto[4] lässt den Benutzer auf Beispielseiten Stellen markieren und einen von drei Regeltypen auswählen. Zum Zerteilen von PH wird der (veraltete) *Java Swing*

[1] Früher: Caesius Software
[2] `http://abrolhos.lbd.dcc.ufmg.br/ debye/`
[3] `http://www.cs.cmu.edu/ rcm/lapis/`
[4] `http://www.dbai.tuwien.ac.at/proj/lixto/`

```
ebaydocument(S, X) ← getDocument(S = $1, X)
ebaydocument(S, X) ← next(_, S), getDocumentOfHref(S, X)
         next(S, X) ← ebaydocument(_, S), subelem(S, (⋆.content, [(href, .∗),
                      (elementtext, (next page))]), X)
       record(S, X) ← ebaydocument(_, S), subelem(S, .table, X)
                      before(S, X, (⋆.tr, [(elementtext, . ∗ Current.∗)]), 0, 100, _, _)
                      after(S, X, (⋆.img, [(src, . ∗ spacer.gif)]), 0, 100, _, _)
      itemdes(S, X) ← record(_, S), subelem(S, (⋆.td. ⋆ .content, [(href, .∗)], X)
        price(S, X) ← record(_, S), subelem(S, (⋆.td, [(elementtext, \var[Y].∗)]), X),
                      isCurrency(Y)
         bids(S, X) ← record(_, S), subelem(S, ⋆.td, X), before(S, X, .td, 0, 30, Y, _),
                      price(_, Y)
         date(S, X) ← record(_, S), subelem(S, ⋆.td, X), notafter(S, X, .td, 100)
     currency(S, X) ← price(_, S), subtext(S, \var[Y], X), isCurrency(Y)
       amount(S, X) ← price(_, S), subtext(S, [0 − 9]+\.[0 − 9]+, X)
```

Abbildung 3.3: Beispiel eines *Elog*-Programms in *Lixto*

Parser verwendet. Der Benutzer navigiert in einem Baum von verschachtelten Regeln – die zugrunde liegende HTML-Seite ist dabei nicht zu sehen [BFG01b]– und kann die Regeln mit ODER-Operatoren erweitern oder mit UND-Operatoren einschränken. Komplizierte Vorgänger- und Nachfolger-Bedingungen können ebenfalls interaktiv mit dem System erzeugt werden, um einen möglichst robusten Wrapper zu erzeugen. Der Benutzer sieht die zugrundeliegende deklarative Sprache *Elog* [BFG01a], ein *Datalog*-Dialekt, nicht im Quelltext. *Lixto* bietet auch reguläre Ausdrücke und XPath-Ausdrücke innerhalb eines *Elog*-Programms.

ParserStudio[5] teilt den Prozess in zwei Produkte: Erstellen der Zerteiler und Nutzung in einer Ablaufumgebung. Die Extraktionsregeln werden per Mausklick aus Beispielen generiert. Es bietet unterschiedliche Module für die Generierung und Verwaltung der Wrapper und kann automatisch auf Webseiten nach Änderungen suchen. *ParserStudio* beherrscht nach eigenen Angaben [Ite02] JavaScript und das Ausfüllen von Formularen. Dies ist ein Vorteil gegenüber den nicht-kommerziellen Systeme. Vermutlich wird dazu eine Instanz des *Internet Explorers* angesteuert.

WebQL[6] stellt das Internet (und zahlreiche andere Datenformate) aus Sicht des Benutzers als „Datenbank“ dar. Es verwendete eine SQL-ähnliche Syntax und ist eine Sprache zur Integration von heterogenen Quellen. Wrapper können über eine SQL-Schnittstelle genutzt werden. Der verwendete PH-Zerteiler ist nicht bekannt. Zur Wrapper-Erstellung steht eine interaktive Entwicklungsoberfläche zur Verfügung.

[5] http://www.itemfield.com/parserstudio.shtml
[6] http://www.caesius.com/webql.html

```
<ST_extract>
ST_extract(String ST_name[], String ST_val[][])
<!-- Start of the repetition -->
<? XG-Iteration-XG ‘‘Start"?>
 <loop> integer row_i = 3, 4
  <loop> integer col_j = 0,1,2
   <rule_exp>
    extract ST_val[row_i,col_j] =
      ~TABLE[2].TR[row_i].TD[col_j].getStoken()
    where ~TABLE[2].TR[1].TD[col_j].getStoken()
          = ST_name[col_j];
   </rule_exp>
  </loop>
 </loop>
</ST_extract>
```

Abbildung 3.4: Beispiel für einen Extraktionsausdruck in *XWrap*

XWrap[7] ist nur online verfügbar, die komplette Schnittstelle zum Benutzer läuft im Browser ab. Für eine gegebene Webseite können semi-automatisch unter Nutzung von Heuristiken einfache Wrapper erzeugt werden, die anschließend interaktiv weiterentwickelt werden. Die dabei intern erzeugten Programme (siehe Abb. 3.4) werden aufbereitet und können anschließend als Java-Quelltext heruntergeladen werden. Zur Interpretation von PH wird *JTidy* verwendet. Die Wrapper geben ihr Ergebnis als XML mit einer dazu passenden DTD aus.

X-Fetch Wrapper[8] bietet als einziges kommerzielles System eine kostenlose Vollversion für wissenschaftlichen oder privaten Einsatz. Das System kann beliebigen Text in XML konvertieren, unterstützt Unicode und liest zahlreiche Formate (EDI, AP-Schnittstelle). Als Extraktionssprache kommt die *Date Extraction Language* (DEL) [del01] mit XML-Syntax zum Einsatz. DEL wurde für den *X-Fetch Wrapper* entwickelt und wird mittlerweile vom W3C standardisiert. Es handelt sich dabei um eine XML-Syntax für reguläre Ausdrücke mit einigen zusätzlichen Befehlen zum Erzeugen eines XML-Dokumentes. Ein spezieller HTML-fähiger Zerteiler wird nicht verwendet.

Extraktionssprachen

Ein Vergleich der Extraktionssprachen (siehe Tabelle 3.1) zeigt, dass jedes System eine eigene Sprache benutzt.

Vergleich mit den Kriterien

Die Tabelle 3.2 vergleicht die vorgestellten Systeme mit den definierten Kriterien. Ein Aussage, in wie weit die Wrapper die Klasse der *atomaren Dienste* beherrschen, kann ohne aufwändige Evaluation nicht gemacht werden. *DEBye* und *X-Fetch* extrahieren Daten direkt aus dem PH-Quelltext. Damit genügen

[7] http://www.cc.gatech.edu/projects/disl/XWRAPElite/
[8] http://www.x-fetch.com/wrapper.html

Name	Extraktionssprache	Modell	Lizenz
DEByE	reguläre Ausdrücke	Text	nicht-kommerziell
Lapis	Text Constraints	Text	nicht-kommerziell
Lixto	Elog	Baum	kommerziell
ParserStudio	Parser Script Language	?	kommerziell
WebQL	WebQL (Web Query Language)	?	kommerziell
XWrap	eigene	Baum	nicht-kommerziell
X-Fetch Wrapper	Del (Data Extraction Language)	Text	kommerziell

Tabelle 3.1: Vergleich Xml-fähiger Systeme nach Lizenz und Extraktionssprache

sie nicht den Kriterien K 1b und K 3b. *LAPIS* und *Lixto* verwenden zwar spezielle Ph-Zerteiler, diese erreichen jedoch vermutlich eine geringere Qualität als ein marktbeherrschender Browser. Einzig *XWrap* verwendet einen speziellen, vergleichsweise modernen Zerteiler für Ph. Dieser wird auf Seite 41 evaluiert. Über die Zerteiler von *ParserStudio* und *WebQL* ist nichts bekannt. Die von den verschiedenen Systemen angebotenen Benutzerschnittstellen sind zumindest interaktiv, aber keine ist vollständig Wysiwyg im Browser realisiert. Eine Webschnittstelle zum Export der Xml-Daten besitzt fast jedes System.

	K 1b	K 2	K 3a
DEBye	-	2	+
LAPIS	-	1	-
Lixto	-	2	-
ParserStudio	?	3	+
WebQL	?	1	+
XWrap	-	2	+
X-Fetch	-	1	+

Tabelle 3.2: Vergleich von Wrapper-Systemen anhand von Kriterien
0 = nicht interaktiv, 1 = interaktiv, 2 = teilweise Wysiwyg, 3 = vollständige Wysiwyg-Erstellung von Wrappern, 4 = (3) im Browser

3.2 PH-Zerteiler

Alle dem Autor bekannten Werkzeuge zur Wandlung von PH in eine Baumstruktur sind im Folgenden aufgelistet. Ein detaillierter Vergleich ausgewählter Zerteiler findet sich in Abschnitt 5.2.

CyberNeko HtmlParser[9] ist eine Entwicklung von *Andy Clark*, aufbauend auf dem flexiblem Rahmenwerk *Xerces Native Interface* (XNI) für den XML-Zerteiler *Xerces 2. CyberNeko* wird als *open source* seit Februar 2002 kontinuierlich weiterentwickelt. Zu den Eigenschaften gehören das Zerteilen von HTML-Fragmenten, das Entfernen von Kommentaren um `<SCRIPT>`-Elemente herum und ausgefeilte Konfigurationsoptionen für die in XML wichtige Behandlung von Entities.

JTidy[10] ist das älteste und verbreitetste Werkzeug zur Wandlung von PH in ein DOM[11] und wurde ursprünglich von *Dave Raggett* als *Tidy*[12] am W3C in C entwickelt und im Jahre 2000 nach Java portiert. Seitdem werden Änderungen an *Tidy* stets in *JTidy* übernommen. Über die Jahre hinweg hat *JTidy* viele Eigenschaften erhalten, die über einen reinen Zerteiler hinausgehen: z. B. Einrücken des erzeugten Quelltextes, Vereinfachen von *Winword 2000*-HTML-Syntax, sowie Aufteilen einer Seite in mehrere. *JTidy* ist bei der Interpretation von PH strenger als die meisten Browser und sieht es als Fehler an, wenn unbekannte Tags vorkommen [htt]. Zwar besitzt es eine Konfigurationsdatei, in der man ihm neue Tags „beibringen“ kann, dies ist aber nicht a priori für alle denkbaren Tags möglich. Nicht in HTML-Kommentaren verstecktem JavaScript-Programmtext zerstört *JTidy*, indem Anführungszeichen als XML-Entities gewandelt werden.

Nazareno[13] verfolgt den Ansatz, die PH-Seite in XML zu überführen und zusätzlich ein XSLT-*Stylesheet* zu erzeugen, die, auf die XML-Datei angewandt, exakt wieder das ursprüngliche PH erzeugt. Ebenfalls zusätzlich werden eine DTD und ein XML-Schema erzeugt, sowie JAXB[14]-Klassen generiert, welche die XML-Serialisierung übernehmen können. Es wurde als *open source* Projekt von *Jesus N. Rodriguez* entwickelt und kann noch keine HTML-Kommentare oder eingebettetes JavaScript behandeln.

Small Html Parser[15] von *Frank Font* legt besonderen Wert auf Geschwindig-

[9] `http://www.apache.org/~andyc/neko/doc/index.html`
[10] `http://sourceforge.net/projects/jtidy/`
[11] Document Object Model
[12] `http://tidy.sourceforge.net/`
[13] `http://www.geocities.com/jesus2nyc/index.html`
[14] Java Architecture for XML Binding erzeugt aus einer DTD Java-Klassen die XML als Objekte darstellen
[15] `http://www.room4me.com/software/SmallHTMLParser.htm`

keit und Speicherbedarf. Er versucht PH als XHTML zu zerteilen.

Swing Html Parser[16] ist Bestandteil des JDK[17] (seit Version 1.2). Die Klasse `javax.swing.text.html.parser.Parser` implementiert einen einfachen DTD-gesteuerten HTML 3.2 Parser, basierend auf *JavaCC*. *Sun* selbst erklärt:

> Unfortunately there are many badly implemented HTML parsers out there, and as a result there are many badly formatted HTML files. This parser attempts to parse most HTML files. This means that the implementation sometimes deviates from the SGML specification in favor of HTML.

TagSoup[18] wurde von *John Cowan* als *open source* entwickelt. Der Zerteiler versucht jedes PH in korrektes XML zu verwandeln ohne jemals einen Syntaxfehler zu melden. Er unterstützt HTML 4.01 und zahlreiche Erweiterungen von *Internet Explorer* und *Netscape 4*, dessen Verhalten er nachahmt.
Der Zerteiler macht folgende Annahmen:

- unbekannte Elemente sind leer
- unbekannte Attribute sind vom Typ `CDATA`
- alle Namen sind kleingeschrieben

TagSoup benutzt eine eigene Schema-Sprache, die in Objekte übersetzt wird und hat noch einige Probleme mit eingebettetem JavaScript-Programmtext.

Zerteiler im Vergleich

Nahezu jedes *open source*-Projekt, das PH einlesen muss, nutzte bislang *JTidy*. In den letzten sechs Monaten (Ende 2002/ Anfang 2003) konnte bei einigen Projekten ein Wechsel zu *CyberNeko* beobachtet werden. Die Tabelle 3.3 vergleicht die PH-Zerteiler hinsichtlich Aktualität und Quelltextgröße. Hier fällt *JTidy* als das mit Abstand umfangreichste Projekt auf.

Name	Ausgabe	letzte Version	Sprache	rlosc
CyberNeko	SAX	03.2003, 0.7.4	Java	3248
Tidy	DOM, HTML 4.01 XML, XHTML	08.2000, tidy4	C	–
JTidy	siehe Tidy	08.2001, r7dev	Java	9879
Nazareno	XML + XSLT	02.2002, 1.0	Java	4957
SmallHTML	Java-Objekte	08.2002	Java	754
Swing HTML Parser	W3C-DOM	09.2002, JDK 1.4.1	Java	–
TagSoup	SAX-Ereignisse	01.2003	Java	2036

Tabelle 3.3: PH-Zerteiler im Vergleichrlosc = real lines of source code

[16] `http://java.sun.com/j2se/1.4.1/docs/api/javax/swing/text/html/parser`
[17] Java Development Kit
[18] `http://mercury.ccil.org/~cowan/XML/tagsoup/`

4 Entwurf

Wie in der Einleitung und im Grundlagen-Kapitel erläutert, dient der Browser in zweierlei Hinsicht als Vorbild für die Wrapper-Erzeugung. Zum einen soll die Benutzerschnittstelle der gewohnten Navigation im Internet mit einem Browser so weit wie möglich gleichen, wodurch ein WYSIWYG-Ansatz motiviert wird. Zum anderen soll die visuelle Darstellung einer Seite im Browser das Vorbild für die Interpretation der PH-Daten durch den Wrapper sein. Dadurch ist z. B. gewährleistet, dass Daten, die im Browser als Tabelle angezeigt werden, auch als syntaktisch repräsentierte Tabelle extrahiert werden können. Für andere Strukturen gilt dies analog. Die extrahierten Daten sollen zur einfachen Integration in bestehende Systeme als XML über eine einfache Webschnittstelle abrufbar sein (siehe S. 8).

Im Folgenden wird die Kernidee des Entwurfs, die Darstellung von HTML in XML und Extraktion daraus, erläutert. Anschließend werden noch offene Entwurfsentscheidungen, um eine WYSIWYG-Schnittstelle zu realisieren, getroffen und begründet. Am Ende dieses Kapitels wird eine Architektur vorgestellt, mit der die Kernidee den Überlegungen entsprechend umgesetzt werden kann.

4.1 Kernidee

Moderne Browser bieten über JavaScript eine DOM[1]-Schnittstelle. Ab Version *dom level 2* kann darüber jedes HTML-Element beliebig modifiziert, entfernt oder hinzugefügt werden, wobei das HTML-Dokument als Baum dargestellt wird. Daraus kann gefolgert werden, dass die interne Darstellung eines Browsers vermutlich ebenfalls eine Baumstruktur ist. Die aus einer PH-Zeichenkette gewonnene Baumstruktur soll zur Weiterverarbeitung gespeichert werden. Dies motiviert XML, als standardisierte und verbreitete Baumbeschreibungssprache, einzusetzen. Für XML existiert auch bereits eine große Zahl von Werkzeugen.

Die zentrale Idee dieser Arbeit ist die Extraktion aus einem XML-Baum, der aus einer PH-Zeichenkette analog zu einem Browser erzeugt wurde.

Zunächst soll der benötige „XML-Baum“ definiert und seine Herkunft geklärt werden.

[1] Document Object Model

Zur einfachen Kennzeichnung wird in dieser Arbeit eine aus PH erzeugte und in XML repräsentierte Baumstruktur mit „XH“ bezeichnet. Die Abkürzung XH steht dabei für „XML-Darstellung aus (Praxis-)HTML“.

Definition: XH ist die aus PH erzeugte und in XML repräsentierte Baumstruktur

Anhand einer Grafik (Abb. 4.1) wird die Rolle von XH erläutert. Zunächst wird der normale Ablauf beschrieben: PH wird vom Browser interpretiert (i) und in eine interne Darstellung abgebildet. Daraus wird durch eine weitere Abbildung r eine visuelle Darstellung erzeugt. Die Abbildungen i und r sind unbekannt. Zur

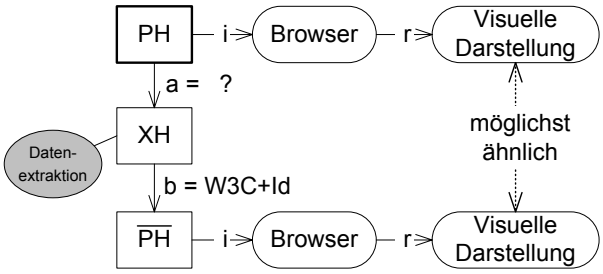


Abbildung 4.1: Zusammenhang zwischen PH, XH und dem Browser

Nutzung im Wrapper findet ein geänderter Ablauf statt. Zunächst wird aus PH durch eine Abbildung a ein gültiges XML-Dokument erzeugt. Dieses wird durch eine Abbildung b wieder in PH verwandelt, wobei b wie folgt definiert ist:

> Für alle Elemente, die im XHTML-Standard definiert sind, spezifiziert das W3C eine Abbildung nach HTML. Dabei wird im wesentlichen jedes Element auf sich selbst abgebildet, bis auf einige wenige Elemente wie `<IMG>`, `<BR>` und `<HR>`, die in HTML nur allein vorkommen dürfen, in XML aber korrekt geklammert sein müssen. Diese werden in die HTML-Form geändert. Weitere Elemente, die in XHTML nicht definiert sind, werden unverändert übernommen.
> Eine solche Abbildung ist im XSLT-Standard bereits definiert. Sie wird in einem XSLT-*Stylesheet* durch die Option `<xsl:output type="html"/>` genutzt.

Das so erhaltene $\overline{\text{PH}}$ wird analog zum ursprünglichen Ablauf vom einem Browser durch i interpretiert und durch r in eine visuelle Darstellung gebracht. Die auf den beiden unterschiedlichen Wegen erhaltenen visuellen Abbildungen sollen sich nun möglichst ähnlich sein.

Wenn die beiden visuellen Darstellungen sich ähneln, muss die Abbildung a semantisch eine ähnliche Abbildung wie i sein, denn die Abbildung b bildet lediglich die XML-Syntax auf HTML-Syntax ab. Da, wie oben erläutert, davon ausgegangen werden kann, dass der Browser intern eine Baumstruktur erzeugt, muss die XH-Darstellung auch dieser Baumstruktur ähnlich sein. Dadurch eignet sich XH gut zur Datenextraktion.

Eigenschaften von XH

Einige Eigenschaften von XH werden nun zusammengefasst:

- Die extrahierten Daten – und das ist entweder das gesamte Dokument oder Teile davon – können durch die die Abbildung b in einem Browser angezeigt werden.
- XH ist eine echte Obermenge von XHTML, da in XH auch Elemente vorkommen dürfen, die nicht in XHTML definiert sind.
- XH ist durch eine einfaches XSLT-*Stylesheet* wieder in PH zurück zu konvertieren, welches wie das ursprüngliche PH im Browser dargestellt wird.

Die Abb. 4.2 zeigt die Zusammenhänge zwischen den verschiedenen Sprachen. SGML[2] ist eine Meta-Sprache mit der die Dokumentsprache HTML definiert wird.

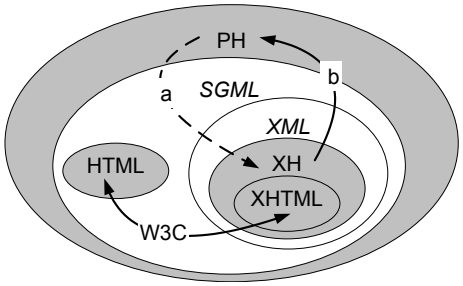


Abbildung 4.2: PH, XML und HTML als Mengen

XML ist selbst wieder eine Meta-Sprache und eine echte Untermenge von SGML. XHTML [XHT02] ist die Reformulierung von HTML 4.0 als Instanz von XML, wobei die Unterschiede nur syntaktischer Natur sind. Das W3C definiert Abbildungen zwischen XHTML und HTML (siehe oben).

Abbildung a von PH nach XH

Wie kann die Abbildung a realisiert werden? Es existieren einige frei verfügbare Komponenten, die PH in XML überführen können. Zur Auswahl, welche dieser Komponenten am Besten für das hier entwickelte System d_2c geeignet ist, wurde ein Benchmark entwickelt (siehe 5.3).

Prinzipiell sind unterschiedliche Interpretationen von verbreiteten Browsern und PH-Zerteiler-Komponenten kein Problem für die Datenextraktion, solange die Abweichungen konsistent sind. Um eine möglichst ähnliche visuelle Darstellung von PH und $\overline{\text{PH}}$ zu erhalten, sollten die Abweichungen möglichst klein sein.

4.2 Entwurfsentscheidungen

Ausgehend von einem XH-Dokument und dem Ziel eine WYSIWYG-Schnittstelle zu realisieren werden nun die noch offenen Entwurfsentscheidungen getroffen.

[2] Standard Generalized Markup Language, ISO-Standard 8879 von 1986

4.2.1 Extraktionssprache

Aus dem als XH vorliegenden PH-Baum sollen die gewünschten Daten extrahiert werden.

Dazu bietet sich die Baumtransformationssprache XSLT[3] an, welche speziell für diesen Zweck konzipiert wurde und wie XML eine große Verbreitung und Werkzeugunterstützung findet. XSLT wird in XML-Syntax notiert.

Optimierungstechniken durch Einsatz von XSLT nutzbar

Durch Verwendung von XSLT sind wohlbekannte Optimierungstechniken einsetzbar (z. B. faule Auswertung [SN03]) wodurch die Ablaufgeschwindigkeit gesteigert werden kann. Da XSLT standardisiert ist, bleiben die systeminternen Schnittstellen beim Austausch des XSLT-Prozessors unberührt. Dadurch kann das System zeitnah von neuen Implementierungen profitieren.

Fazit: XSLT ist ein verbreiteter Standard speziell für die Transformation von – und damit Extraktion aus – Bäumen. Durch Nutzung von XSLT können Optimierungstechniken genutzt werden.

4.2.2 Erzeugen des Xslt-Stylesheets

WYSIWYG-Erstellung

Woher kommt das extrahierende XSLT-*Stylesheet*? Obwohl es zahlreiche Werkzeuge zum Erstellen und Bearbeiten von XSLT gibt, ist das manuelle Erstellen oder Nachbearbeiten mühsam. Die XSLT-*Stylesheet*-Erstellung soll daher über eine WYSIWYG-Schnittstelle im Browser erfolgen, so dass der Benutzer nicht mit XSLT im Quelltext konfrontiert wird.

Erstellung des Xslt-*Stylesheets* als Lernaufgabe

Zunächst kann die XSLT-*Stylesheet*-Erstellung als Lernaufgabe formuliert werden: Eingabedaten sind eine Menge von *gleichartigen* Seiten (Definition in 2.3), in denen jeweils einige Elemente selektiert sind. Gesucht ist *ein* XSLT-*Stylesheet*, welches aus allen gegebenen Seiten genau die selektierten Elemente ausschneidet. Wenn die Seiten zu unterschiedlich sind, existiert eine solches *Stylesheet* nicht und das Lernproblem ist damit unlösbar.

Lernproblem ist ein Selektionsproblem

Ein das Lernproblem lösendes XSLT-*Stylesheet*, sofern es existiert, besitzt für jedes zu extrahierende Element einen XPath-Ausdruck, der es im Baum eindeutig referenziert. Es ist zu erwarten, dass ein solches *Stylesheet* auch auf zukünftigen, ungesehenen Seiten desselben datenerzeugenden Prozesses, die Extraktion von gewünschten Elementen leistet. Hierbei können problemlos absolute Pfade verwendet werden, da relative keinen Vorteil bringen: Strukturelle Änderungen der Seite ziehen in jedem Fall ein manuelles Überprüfen des Wrappers nach sich, um semantische Fehler auszuschließen. Diese Überprüfung soll visuell erfolgen.

Ein extrahierendes XSLT-*Stylesheet* ist durch die XPath-Ausdrücke der zu extrahierenden Elemente gekennzeichnet. Damit wird das Problem der Erstellung eines XSLT-*Stylesheet* auf das Problem der Auswahl von HTML-Elementen reduziert. Es genügt eine Selektion für eine einzige Seite.

[3] eXtensible Stylesheet Language Transformations

WYSIWYG-Selektion im Browser

Der Benutzer soll möglichst ohne Kontextwechsel „nebenbei“ einen Wrapper erzeugen können. Da der Benutzer zum Navigieren und Betrachten von PH einen Browser benutzt, ist es nahe liegend, auch einen Browser zur Selektion von Elementen zu benutzen.

Instrumentalisierte Webseiten als Selektionswerkzeug

Um eine WYSIWYG-Oberfläche zur XH-Element-Selektion im Browser zu realisieren, soll die Original-PH-Seite instrumentalisiert werden – ohne die Seite visuell zu verändern. Mit „instrumentalisieren“ ist die Anreicherung einer PH-Seite mit zusätzlichen Daten – HTML-Elementen und JavaScript – gemeint.

4.2.3 Interaktion mit einem Browser

Filternder Proxy zur transparenten Manipulation von Webseiten

Der Inhalt von PH-Seiten, die der Benutzer anfordert, sollen auf dem Weg vom Webserver zum Browser verändern werden, um eine Benutzerschnittstelle zu realisieren.

Um den gesamten HTTP-Verkehr des Browser behandeln zu können, kann das System sich entweder als Webserver gegenüber dem Browser darstellen oder als HTTP-Proxyserver fungieren. Um als Webserver auftreten zu können, muss *jede* PH-Seite instrumentalisiert werden. Da dieser Prozess jedoch potentiell fehleranfällig ist, sollen so wenig Seiten wie möglich instrumentalisiert werden. Ein bereits existierender Mechanismus für dieses Problem sind HTTP-Proxys. Ein Browser schickt bei aktiviertem Proxyserver seinen gesamten HTTP-Verkehr über diesen. Dazu muss er einmal in den Browser-Einstellungen eingetragen werden. Proxys werden meist nur zur Zwischenspeicherung von Dokumenten benutzt, können aber technisch gesehen beliebiges mit dem Inhalt tun, ihn also auch verändern. Ein solcher verändernder Proxy wird oft als „Filternder Proxy“ bezeichnet. Der Proxy kann individuell für jede HTTP-Anfrage entscheiden, ob sie verändert werden soll. So können nur die Seiten, aus denen Daten extrahiert werden sollen, instrumentalisiert werden.

Das System soll den Benutzer in seinem Navigationsverhalten beobachten, um dieses Verhalten später simulieren zu können. Als Erweiterung des Systems ist geplant (siehe Seite 83), aus diesen Daten auch automatisch Navigationsregeln zu lernen.

Bookmarklets als Bedienelemente

Da keine weiteren Bedienelemente auf der Seite eingebettet werden sollen, erfolgt die Kommunikation mit dem HTTP-Proxy über die von *Steve Kangas* beschriebenen *Bookmarklets*[4]. Das sind JavaScript-Aufrufe welche als Lesezeichen im Browser gespeichert werden. Da alle verbreiteten Browser JavaScript-Programme global ausführen, ist es möglich, über ein solches Lesezeichen ein Skript innerhalb der gerade geladenen Seite aufzurufen. Die meisten Browser verfügen über eine Sorte besonders wichtiger Lesezeichen, die permanent in einer Menüleiste angezeigt werden (siehe Abb. 4.3). Eine Alternative ist ein in JavaScript programmiertes Kontextmenü. Dieses kann aber nur in instrumentalisierten Seiten angezeigt werden.

[4] `http://www.bookmarklets.com`

Abbildung 4.3: Darstellung der *Bookmarklets* in einem Browser

Weitere Interaktionsmöglichkeiten

Der Status des Proxys kann dem Benutzer durch Manipulation der PH-Seite präsentiert werden. Am Anfang einer PH-Datei können eine Tabelle, ein `IFRAME` oder andere Elemente eingefügt werden. Die in d_2c realisierte Möglichkeit ist der Einbau einer Statusanzeige im Titel, wodurch die im Browser-Fenster sichtbare Seite nicht verändert wird.

Fazit: Das Entwicklungssystem von d_2c wird als „Filternder Proxy“ realisiert, um möglichst wenig Seiten instrumentalisieren zu müssen, da dieser Prozess fehleranfällig ist, und trotzdem den gesamten HTTP-Verkehr des Browsers behandeln zu können. Durch vom Proxy instrumentalisierte Seiten kann eine WYSIWYG-Schnittstelle zur Element-Selektion realisiert werden. *Bookmarklets* ermöglichen das Anzeigen von grafischen Bedienelementen für eine PH-Seite ohne in den PH-Text einzugreifen.

4.2.4 Umgang mit veränderten Quellseiten

Wie in Abschnitt 2.4.4 beschrieben, verändern sich Webseiten auch *strukturell*. Dadurch zeigen die XPath[5]-Ausdrücke potentiell auf andere – und damit falsche – Elemente. Dies gilt gleichermaßen für absolute und relative XPath-Ausdrücke und jede andere Art der Referenzierung innerhalb der PH-Seite. Ein Wrapper sollte manuell überprüft werden; ein System kann lediglich einen Vorschlag für einen reparierten Wrapper liefern (siehe auch 8.2.5), denn es kann die semantische Auswahl des Benutzers nicht nachvollziehen. Wrapper sollten vergleichsweise leicht „zerbrechen“, das heißt aufgrund von geänderten Seiten nicht mehr extrahieren. d_2c soll keine *robusten* Wrapper (2.4.4) erzeugen. Statt dessen soll die Wrapper-Erstellung so weit wie möglich vereinfacht werden, wodurch bei einer veränderten Quellseite rasch ein neuer Wrapper für die veränderte Seite erstellt werden kann. Die Gefahr durch Reparatur-Heuristiken falsche Daten zu extrahieren wird umgangen.

Änderungen auf Webseiten erkennen

Zu den *strukturellen* Elementen einer Seite (siehe 2.3) können XPath-Ausdrücke erzeugt werden. Bei erneutem Analysieren der Seite können diese als Rückversicherung benutzt werden; sie dürfen sich nicht verändert haben. Diese Methode ist in d_2c noch nicht implementiert.

[5] XML Path Language, adressiert XML-Elemente, ähnlich wie Pfade dies in einem Dateisystem tun, ist aber ausdrucksmächtiger

Fazit: Sobald sich eine Seite strukturell ändert, geht die Auswahl-Semantik des Benutzers potentiell verloren und ein erneutes Selektieren ist erforderlich. Aus diesem Grund muss ein Wrapper strukturelle Änderungen an der zugrunde liegenden Quellseite erkennen.

4.3 Architektur

Zwei Schichten

Die Architektur des Gesamtsystems lässt sich, wie in [GLdSRN00] in zwei Teile zerlegen: Die Erstellung eines extrahierenden Wrappers erfolgt mit dem *Entwicklungssystem* und seine Anwendung innerhalb des *Produktionssystems*.

Modularisierung

Diese beiden Schichten lassen sich wiederum, den Überlegungen dieses und der vorangehenden Kapitel folgend, in Module aufteilen. Ein Modul soll einfach durch eine andere Implementierung austauschbar sein. Das *Produktionssystem* als Wrapper-Ablaufumgebung lässt sich Abschnitt 2.4.3 folgend in drei Module untergliedern:

surf	Führt HTTP-Anfragen aus.
clean	Zerteilt PH-Zeichenketten in einen wohlgeformten XH-Baum.
extract	Extrahiert Elemente aus dem XH-Baum mit XSLT

Das *Entwicklungssystem* zur Wrapper-Erstellung besteht ebenfalls aus drei Modulen:

proxy	HTTP-Proxy, filtert PH-Seiten, zeichnet HTTP-Verkehr auf
learn	Erzeugt ein XSLT-*Stylesheet*, das aus einem XH-Baum die selektierten PH-Elemente extrahiert
ui	Instrumentalisiert PH-Seiten

Ablaufgeschwindigkeit

Da die Erstellung interaktiv mit dem Benutzer erfolgt, werden dort nur in der BenutzerSchnittstelle selbst schnelle Antwortzeiten gefordert. Das komplette Entwicklungssystem darf geringeren Geschwindigkeitsansprüchen genügen. Anders beim Produktionssystem: Da es auch von anderen Systemen genutzt wird, welche zum Teil mehrere Wrapper nutzen, um Daten zu aggregieren und weiter zu verarbeiten, sollen die Wrapper möglichst schnell ausgeführt werden.

4.4 Produktionssystem zur Wrapper-Nutzung

Das Produktionssystem kann eine PH-Seite von einem Webserver holen, sie ähnlich einem Browser in eine Baumstruktur zerteilen und mit einer gegebenen XSL-Transformation die gewünschte Extraktion durchführen.

Ein typischer Ablauf wird in Abb. 4.4 dargestellt. Zunächst wird das Produktionssystem mit einem bestehenden Auftrag (*Job*) aufgerufen (1). Ein Auftrag ist eine HTTP-Anfrage (GET oder POST inklusive der Parameter) und eine dazugehörige XSL-Transformation. Die gespeicherte HTTP-Anfrage wird dem Modul *surf* übergeben, das die Anfrage erneut ausführt (2). Der vom Webserver erhaltene PH-Text (3) wird vom Modul *clean* zu XH aufbereitet und an das Modul

extract übergeben (4). Das XH-Dokument wird mit der gegebenen Transformation gewandelt (5) und das Ergebnis der Extraktion als XML in der HTTP-Antwort zurück an den Aufrufer gesendet (6). Der Aufrufer ist beim Erstellen des Wrappers das Modul *proxy* des Entwicklungssystems, später in der Nutzung sind es externe Komponenten.

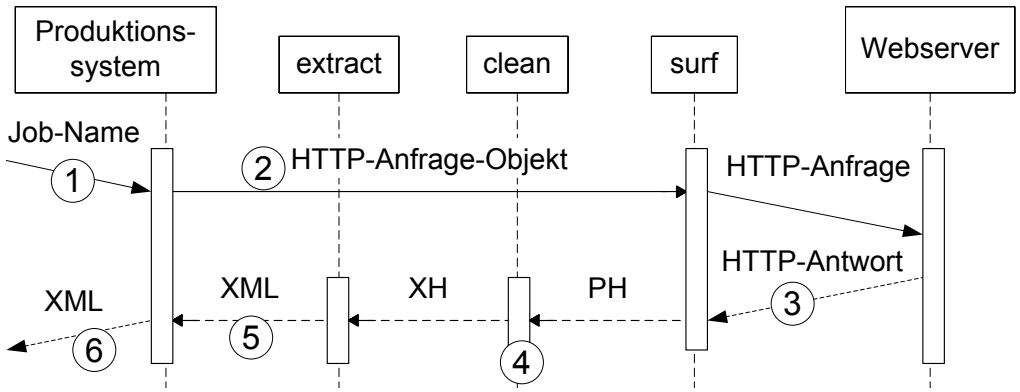


Abbildung 4.4: Ablauf eines Auftrags durch die Module

Die drei Module der Transformationskette (*surf*, *clean*, *extract*) werden im Folgenden näher beschrieben.

4.4.1 Modul surf

Diese Komponente muss die mit dem HTTP-Proxy aufgezeichneten HTTP-Anfragen abspielen können. Dies soll möglichst schnell und robust erfolgen. Das Modul *surf* soll als ein Teil des HTTP-Proxys benutzt werden, dadurch ist ein konsistenter Umgang mit HTTP-Anfragen und -Antworten möglich. Gleichzeitig soll die HTTP-Komponente austauschbar bleiben.

Beim Abspielen eines Wrappers lädt das Modul *surf* die zu extrahierende Seite vom Webserver. Eine aufgezeichnete HTTP-Anfrage ist beim Abspielen auf Wunsch zu verändern, damit Anfrageparameter zur Laufzeit des Wrappers neu gesetzt werden können. Dies ist für datensatzorientierte Extraktion notwendig.

Das Modul muss Anfragen wie ein Browser stellen, sich demnach auch wie ein solcher gegenüber dem Server identifizieren und die gebräuchlichen Nutzungen von HTTP im Browser-Umfeld gut beherrschen.

Kodierungen

Kodierungen in HTTP und HTML sind kompliziert: Die beiden Standards ermöglichen viele, auch widersprüchliche Angaben über die verwendete Kodierung (siehe Seite 69). Als Strategie werden die unterschiedlichen Ausgaben der verschiedenen Webserver so früh wie möglich, also bereits hier im Modul *surf*, in eine einheitliche Kodierung gebracht.

Die entsprechenden Kodierungen müssen bei der Manipulation von XH-Dokumenten und der Transformation mit XSLT beachtet werden: Eingefügte Elemente müssen der umgebenden Kodierung entsprechen.

4.4.2 Modul clean

Das Modul *clean* kapselt einen fehlertoleranten PH-Zerteiler. Es nimmt vom Modul *surf* eine PH-Zeichenkette entgegen und liefert dem Extraktionsmodul ein daraus erzeugtes Dokumentmodell als Baumstruktur zurück.

Da es bedeutend weniger Typen von Webservern als Webseiten erzeugende Prozesse (manuell, zahlreiche Schablonensysteme, zahlreiche Editoren) gibt, ist die Auswahl einer Basiskomponente für das Modul *clean* entscheidend dafür verantwortlich, ob die von d_2c erzeugten Wrapper Daten analog zu einem Browser extrahieren können.

Der Ansatz von [Myl01] lautete:

> ... much of the HTML content on the Web today is ill-formed because it does not conform to HTML specifications. Therefore, the first step in data extraction is to translate the content to a well-formed XML syntax because this helps in subsequent data extraction steps.

Analog dazu ist die Wandlung von HTML zu XML – genauer PH zu XH – der entscheidende Schritt zur automatisierten Verarbeitung von syntaktisch nicht korrekten Inhalten. Allerdings gibt es für die komplexe Aufgabe der Wandlung von PH nach XH keine bestehenden Standards. Die Auswahl eines geeigneten PH-Zerteilers wird mit Hilfe eines speziell für diese Arbeit entwickelten Benchmarks durchgeführt (Seite 37).

4.4.3 Modul extract

Dieses Modul kann aufgrund der Vorarbeiten stark von bestehenden Standards profitieren: Mit dem Entwicklungssystem wurde interaktiv ein XSLT-*Stylesheet* erstellt, das den vom Modul *clean* gelieferten XML-Baum transformieren muss. Für die Transformation von XML mit XSLT existieren zahlreiche Implementierungen. Durch das *Stylesheet* wird ein XML-Dokument erzeugt und zurückgeliefert, in welchem die vom Benutzer selektierten Elemente verpackt sind.

4.5 Entwicklungsumgebung zur Wrapper-Erstellung

Die Aufträge werden mit dem Entwicklungssystem erstellt, welches aus einem Hauptmodul (*proxy*) und zwei Hilfsmodulen (*learn*, *ui*) besteht. Zunächst wird das Entwicklungssystem als Ganzes beschrieben.

Erstellung eines Wrappers

Die Erstellung (siehe Abb. 4.5) erfolgt in den zwei Phasen *Navigation* (A) und *Selektion* (1-8).

Phase 1: Navigation im Browser

Zunächst ruft der Benutzer Webseiten wie gewöhnlich auf. Das Modul *proxy* nutzt dazu das Modul *surf* um die HTTP-Anfragen des Browsers weiterzuleiten. Die empfangenen HTTP-Antworten werden nur in den Headern modifiziert (wegen Zwischenspeicherung, siehe 6.2.1) und zurückgeliefert. Dies kann sich beliebig oft wiederholen, bis der Benutzer eine Seite findet, für die er einen Wrapper erzeugen möchte.

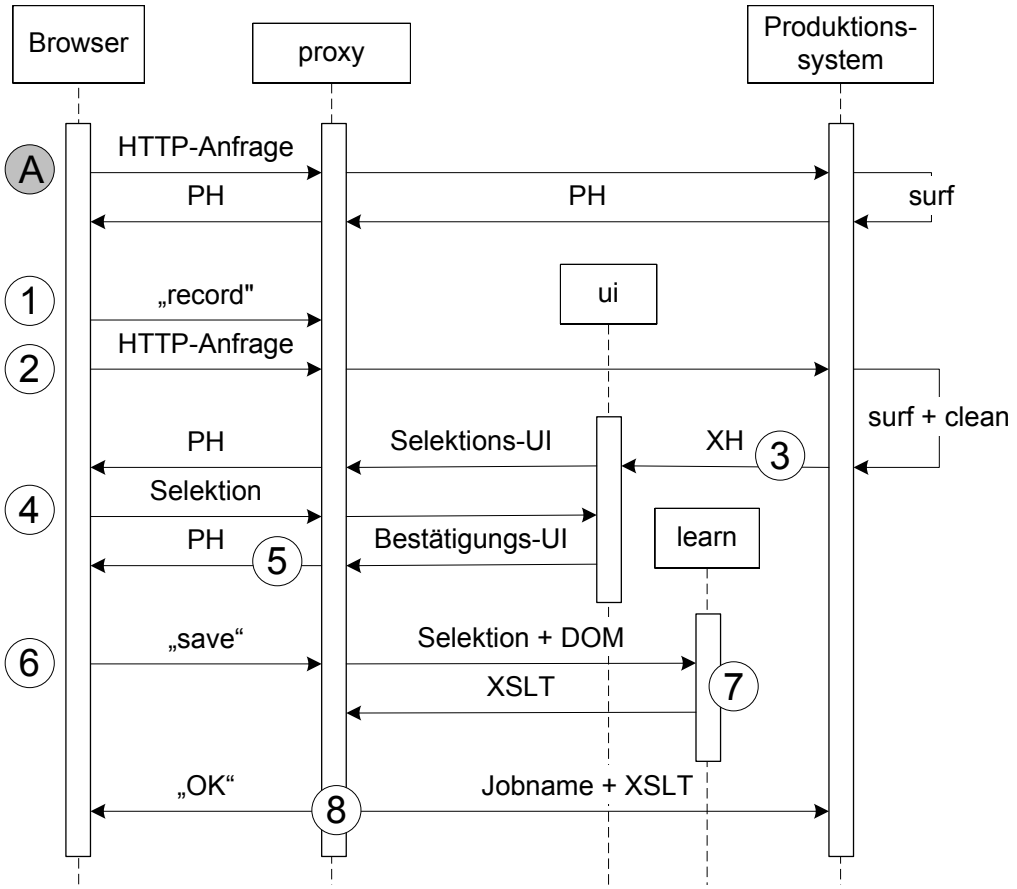


Abbildung 4.5: Erstellen eines Auftrags

Phase 2: Selektion im Browser

Dazu sendet er über ein *Bookmarklet* den Befehl `record` an den Proxy (1). Die nächste HTTP-Anfrage wird als Extraktionsquelle betrachtet und dem Produktionssystem zum Herunterladen des PH-Textes und Aufbereiten als XH-Baum übergeben (2). Der erhaltene XH-Baum wird vom Modul *ui* in eine PH-Selektionsoberfläche verwandelt, die über den Proxy an den Browser zurück gesendet wird (3). Der Benutzer wählt im Browser die Elemente aus, die ihn interessieren. Dabei erfolgt keine Kommunikation mit dem Proxy (4). Die komplette Selektion wird über das *Bookmarklet* `select` an den Proxy gesendet. Dieser fragt beim Modul *ui* eine Bestätigung der Selektion an. Die gewählten Elemente erscheinen dem Benutzer jetzt z. B. grün eingefärbt (5). Der Benutzer bestätigt die Selektion mit dem *Bookmarklet* `save` und gibt einen Namen an (6). Die Selektion und das zugrunde liegende XH-Dokument gehen an das Modul *learn*. Dieses erzeugt ein extrahierendes XSLT-*Stylesheet* (7). Der Proxy meldet dem Benutzer eine Bestätigung und veranlasst das Produktionssystem, den neu erstellen Wrapper unter einem gegebenen Namen zu speichern (8).

Kopplung von sichtbarem PH und DOM-Repräsentation

Da der Benutzer aus dem HTML-Baum Elemente im Browser selektieren soll, muss eine Verbindung zwischen der im Browser als PH sichtbaren XH-Darstellung und der internen DOM-Repräsentation des selben XH-Baums bestehen.

Der Benutzer erhält dazu die ursprüngliche HTML-Seite, wobei die Elemente mit eindeutigen Nummern (als nicht-sichtbare Attribute) angereichert wurden. Zusätzlich enthält jedes Element einen JavaScript-Aufruf, der über dynamische CSS-Anweisungen – d.h. CSS-Anweisungen werden über JavaScript manipuliert – die selektierten Elemente visuell hervorhebt. Eine Selektion auf dem Baum wird als Folge von kommagetrennten Elementnummern wieder an den Proxy übertragen.

4.5.1 Modul proxy

Der Proxy in d_2c muss zwei Aufgaben erfüllen:

- Erstens das Aufzeichnen von HTTP-Anfragen, so dass sie wieder abgespielt werden können.
- Zweitens werden die empfangenen HTTP-Antworten gefiltert, um in die PH-Seiten die BenutzerSchnittstelle zur Proxysteuerung und Elementselektion einzubauen.

Das Modul *proxy* stellt die zentrale Komponente dar und nutzt das Produktionssystem, um eine Seite zu laden und in XH zu überführen. Anschließend wird die Seite vom Modul *ui* verändert und an den Browser ausgeliefert.

Zunächst wurde auch die Möglichkeit, einen fertigen filternden Proxy zu verwenden betrachtet. Als einziges relevantes *open source*-Projekt dafür stellt sich *Muffin*[6] heraus. Es wird aber nicht mehr aktiv weiterentwickelt – die letzte Version ist vom April 2000.

Um HTTP-Anfragen wieder abspielen zu können, bietet sich an, das Modul *surf* als Bestandteil des HTTP-Proxys zu verwenden. Dazu wird eine wesentliche zweite Komponente, ein Webserver, benötigt. Der Proxy selbst ist Brücke zwischen HTTP-Server und HTTP-Client. Er leitet beim Server eingehende Anfragen an den HTTP-Client weiter. Die HTTP-Antwort wird umgeschrieben und über den Server zurück an den Aufrufer gesendet. In Abb. 4.6 ist der Ablauf dargestellt.

Es bleibt ein Proxy-Kern zu erstellen, das ist der zwischen HTTP-Server und dem Modul *surf* vermittelnde Teil. Zur Entkoppelung von Server und Client wird eine problemspezifisch manipulierbare Zwischendatenstruktur verwendet. So kann bei Bedarf ein anderer HTTP-Client verwendet werden – eine Strategie, die sich im Projektverlauf bewährt hat.

So wenig wie möglich filtern

Durch den HTTP-Proxy laufen alle HTTP-Anfragen und -Antworten. Nur HTTP-Antworten auf denen die Schnittstelle zur Wrapper-Erstellung laufen soll, müssen in XH überführt und instrumentalisiert werden. Durch die Injektion von JavaScript-Programmtext ändert sich die Struktur des Dokuments leicht, wodurch einige wenige JavaScript-Funktionen der Original-Seite dann nicht mehr

[6] `http://muffin.doit.org/`

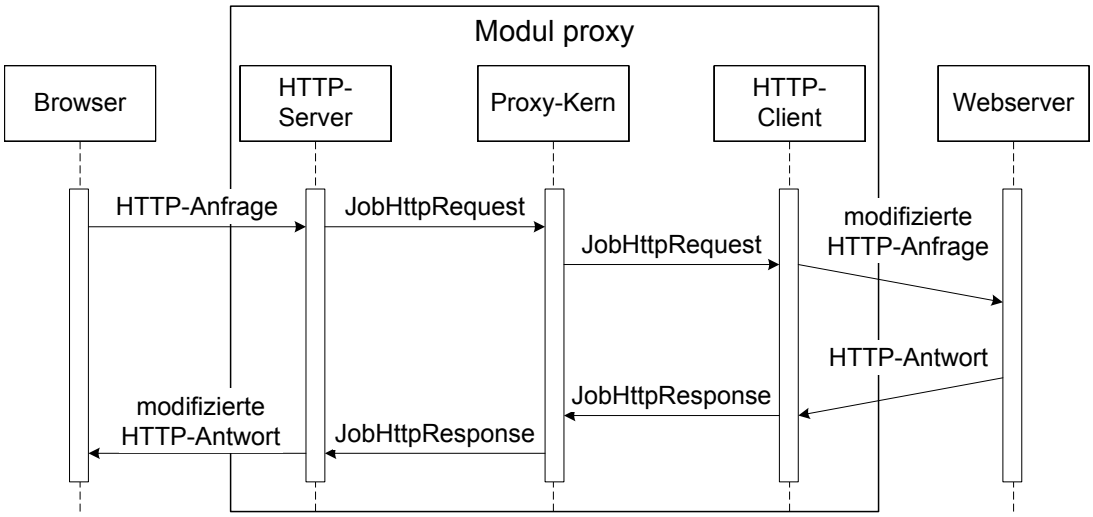


Abbildung 4.6: Einbettung des Moduls *proxy*

korrekt ablaufen. Es sind also so wenig Seiten wie möglich zu instrumentalisieren.

Steuerung durch Spezial-URLs

Der Proxy soll über den Browser steuerbar sein. Notwendige Kommandos sind:

- Wechsel in den Aufnahme-Modus
- Bestätigen und Speichern der Selektion als neuen Wrapper
- Abfragen eines Wrappers

Dazu können verschiedene Ports für die verschiedenen Befehle benutzt werden, was aber nicht besonders flexibel ist und einen hohen Administrationsaufwand bei der Konfiguration einer *Firewall* sein kann. Aus diesem Grund werden spezielle URLs verwendet. Für diese URLs können keine Webseiten abgerufen werden, da sie als Steuerinformationen interpretiert werden. Da noch nicht alle *Top Level Domains* (TLD) vergeben sind, kann davon ausgegangen werden, dass sich z. B. hinter der Endung „`.d2c`“ keine Webseite befindet. Der Proxy kann durch den Aufruf von URLs der Bauart „*Befehlsname*`.action.d2c`“ im Browser gesteuert werden, ohne den Benutzer bei der Auswahl von zu betrachtenden Webseiten einzuschränken.

4.5.2 Modul learn

Dieses Modul erzeugt aus einem vom Benutzer annotierten Dokument ein extrahierendes XSLT-*Stylesheet*, welches aus dem DOM genau die selektierten Elemente zurückliefert. Diese Komponente muss selbst geschrieben werden, da es sich um ein sehr spezifisches Problem handelt, welches nicht von bestehenden Komponenten abgedeckt wird. Es kommt in diesem Modul, anders als der Name suggeriert, in der aktuellen Implementierung kein Lernverfahren zum Einsatz.

Lernen eines XSLT-*Stylesheets*? Selektion genügt

Der Benutzer selektiert auf einer instrumentalisierten HTML-Seite die für ihn gewünschten Elemente. *Die Granularität der Auswahl ist genau die der* HTML*-Elementhierarchie, welche direkt in XPath-Ausdrücke überführt werden kann.* Dadurch stellt sich das Lernproblem als ein Selektionsproblem, da lediglich für jedes Element genau eine Extraktionsregel durch einen eindeutigen Pfad im Dokumentbaum (XPath) angegeben werden muss.

Einfache Konzepte genügen

Der Benutzer wählt manuell genau die Elemente einzeln aus, die er extrahieren möchte. Komplizierte Konzepte, wie z. B. „die Spalte einer Tabelle“, „die ersten n Tabellenzeilen“ oder „alle Tabellenzeilen“ müssten einzeln in der Selektionsschnittstelle auswählbar sein, damit ein solches Konzept mit Sicherheit erkannt wird. Das geht aber nur, wenn der Benutzer es explizit angibt (z. B. in einem Kontextmenü der rechten Maustaste). Automatisch könnte zwar nach einigen markierten Tabellenzellen ein Vorschlag für ein solches Konzept gemacht werden, aber es ist zweifelhaft, ob dies dem Benutzer wirklich Mausklicks erspart, da der Automatismus sich leicht irren kann. Diese Konzepte betreffen allerdings nur die Selektion, das Erzeugen des XSLT-*Stylesheets* bleibt davon unberührt. Bei einer Tabelle, die einmal 3 oder auch 4 Spalten enthält, wählt der Benutzer wahrscheinlich die ganze Zeile aus (durch mehrfache Selektion eines der Zeilenelemente), da all diese Daten dynamisch sind.

4.5.3 Modul ui

Die Benutzerschnittstelle lässt sich nicht von einem anderen System übernehmen und muss daher selbst geschrieben werden. Wie in Abschnitt 4.2.2 ausgeführt, wird das Problem der XSLT-*Stylesheet*-Erstellung auf eine Elementselektion reduziert. Es sollen die zu bearbeitenden Daten (bestehende PH-Seite) so manipuliert werden, dass sie das gleiche Erscheinungsbild aufweisen und zusätzlich die Funktionalität der Elementselektion bieten. Es soll dadurch eine reaktive und intuitive Schnittstelle im Browser realisiert werden, in der der Benutzer Elemente durch Anklicken markieren kann. Zusätzlich muss der Proxy mit Befehlen gesteuert werden.

Es wurden folgende Technologien verwendet:

CSS parametrisiert HTML-Elemente zur Visualisierung der Selektion

JavaScript (DOM-level-2) ist für eine interaktive, reaktive Benutzerschnittstelle unverzichtbar. Andernfalls müsste nach jedem Mausklick des Anwenders auf eine HTTP-Antwort vom Server gewartet werden. Selbst wenn der Server auf der selben Maschine läuft wie der Browser, muss dieser jedes Mal eine neue Seite laden und darstellen – das wäre zu langsam.

Dom4j als serverseitiges Dokumentmodell für das zu manipulierende XH-Dokument

Selektion im Browser

Selektierbare Elemente sind auch extrahierbar. Nicht alle HTML-Elemente erlauben `onClick-Handler` und manche sind für eine Extraktion nicht interessant, da sie keine Daten tragen (z. B. horizontale Trennlinien `<HR>`).

Durch folgendes Verfahren ist es möglich, mehr Elemente selektierbar zu machen, als per Mausklick zu aktivieren sind:

- Jeder Klick auf ein Element selektiert es.
- Wenn es schon selektiert war, wird es deselektiert (und ebenso alle Geschwisterelemente im Dokumentbaum) und sein Vaterelement selektiert. Es ist nicht notwendig, dass das Vaterelement selbst direkt anklickbar ist. Die Selektion„wandert“ also bei wiederholtem Klicken auf ein Element nach „oben“ im HTML-Elementbaum.
- Ist die Wurzel selektiert, so ist nach dem nächsten Klick kein Element mehr selektiert.

Ein Beispiel für einen solchen Ablauf findet sich in Abb. 7.3 auf Seite 73.

5 Wiederverwendung

In der Architektur des Gesamtsystems wurden einige Module identifiziert, bei denen einen bestehende Implementierung verwendet werden soll. Dazu gehören ein HTTP-Client und ein Server, die zusammen die Basiskomponenten des Proxys bilden. Zur Repräsentation und Manipulation von XML-Dokumenten wird ein DOM-Paket ausgewählt. Im Kapitel „Entwurf“ wurde die Bedeutung eines fehlertoleranten PH-Zerteilers motiviert. Im Kapitel „Stand der Technik“ wurden solche Zerteiler vorgestellt und drei mögliche Kandidaten identifiziert. Zur Auswahl des fehlertoleranten PH-Zerteilers, der PH am ähnlichsten zu einem Browser zerteilt und aus möglichst jeder Eingabe wohlgeformtes XML erzeugt, wird ein neuer Benchmark entwickelt.

5.1 Auswahl einer Basiskomponente für das Modul surf

Evalution „schlanker“ HTTP-Clients

Das Modul *surf* soll per HTTP mit Webservern kommunizieren und die HTTP-Nachrichten auch modifizieren und weiterleiten können, damit das Modul auch in einem Proxy eingesetzt werden kann. Um an d_2c anpassbar zu sein, werden nur *open source*-Entwicklungen betrachtet. Zunächst werden drei „schlanke“, reine HTTP-Clients evaluiert:

URLConnection ist eine Klasse des *Java Development Kit* (JDK). Sie liefert bei HTTP-Antwort-Codes größer als 400 stets einen Fehler; Cookies werden gar nicht behandelt und der Quelltext ist nicht frei verfügbar. Daher diese Klasse nicht geeignet um mit möglichst jedem Webserver kommunizieren zu können.

HttpClient 0.3-3 wurde von der Schweizer Firma *Innovation* programmiert; die Weiterentwicklung wurde eingestellt.

Dieser Client folgt automatisch den HTTP-Weiterleitungsnachrichten (302). Dadurch lädt der Browser die Daten von der neuen, weitergeleiteten URL, löst relative Links aber relativ zur ursprünglichen URL auf – was zu einer Fehlermeldung im Browser führt. Daher sollte diese Komponente nicht verwendet werden.

HttpClient 2.0 Alpha 2 wird von einer aktiven Entwicklergemeinde im Rahmen eines *Apache Jakarta Commons*[1]-Projektes gepflegt und weiterentwickelt. Für auftretende Fehler wird meist rasch ein *Patch* auf den projekteigenen Mailinglisten veröffentlicht.

[1] `http://jakarta.apache.org/commons/`

Komponente	Lizenz	Modell	rlosc
httpUnit	MIT (public domain)	HTTP-Protokoll, -Anfragen und -Antworten	7000
htmlUnit	Apache	Dokument, Seiten, Formulare, Tabellen	10700
Zum Vergleich: Jakarta HttpClient	Apache	HTTP-Protokoll	7802

Legende: Quelltextmenge in *real lines of source code* (rlosc).

Tabelle 5.1: Vergleich von HTTP-Clients

Vergleich von umfangreicheren Komponenten

Umfangreichere Komponenten, die für das automatische Testen von Webseiten entwickelt wurden, bieten erweiterte Browser-Funktionalitäten, wie z. B. JavaScript und *Frames*. Die beiden *open source*-Projekte *httpUnit*[2] und *htmlUnit*[3] stammen aus diesem Bereich und nutzen intern den *Jakarta Commons HttpClient*. Ein Vergleich (5.1) zeigt, dass *httpUnit* aufgrund des verwendeten HTTP-näheren Modells eher in Frage kommt.

httpUnit und htmlUnit ungeeignet

httpUnit 1.5.1 scheiterte auf den Seiten `www.web.de` und `www.amazon.de` an JavaScript-Fehlern. Bei anderen Seiten lädt *httpUnit* eigenmächtig alle Seiten eines *Framesets* und zerstört dadurch die Synchronisation zum Browser. Dieser kann auf eine HTTP-Anfrage nur eine HTTP-Antwort bekommen. *httpUnit* ist für einen Einsatz in einem HTTP-Proxy daher ungeeignet.

Auch *htmlUnit 1.2.1* ist für eine Nutzung in einem HTTP-Proxy nicht geeignet, da es nach eigenen Angaben noch weiter vom HTTP-Protokoll abstrahiert. Damit können die HTTP-Nachrichten nicht mehr manipuliert und weitergeleitet werden.

Fazit: Die beiden größeren Komponenten erwiesen sich als zu stark vom HTTP-Protokoll entfernt und zu stark automatisiert, um sie als Teil eines HTTP-Proxys zu nutzen. Der in d_2c verwendete *Jakarta HttpClient* ist überschaubar genug, um ihn in das Projekt einzupassen und leistungsfähig genug, um mit fast jedem Webserver zu kommunizieren.

5.2 Auswahl eines fehlertoleranten PH-Zerteilers

Im folgenden wird ein Benchmark entwickelt, der die XH-Bäume von freien Zerteiler-Komponenten syntaktisch analysiert und mit den Syntaxbäumen eines Browsers vergleicht. Dieser Aufwand, um die optimale Komponente für das Modul *clean* zu finden, ist nötig, da der PH-Zerteiler maßgeblich entscheidet, ob d_2c PH wie ein Browser zerteilen kann.

[2] `http://sourceforge.net/projects/httpunit`
[3] `http://sourceforge.net/projects/htmlunit`

Internet Explorer nicht verwendbar

Es liegt Nahe, dass hier der Zerteiler des Browsers mit dem größten Marktanteil (*Internet Explorer*) verwendet werden sollte. Eine Möglichkeit zur Ansteuerung über JavaScript (siehe Abschnitt 5.3.4) ist für ein Produktionssystem nicht geeignet, da ein Browser dabei vergleichsweise leicht „abstürzen“ kann. Der *Internet Explorer* kann unter *Windows* als COM-Objekt[4] angesteuert werden. Die visuelle Darstellung entspricht dabei nicht der programmatisch erhaltenen Baumstruktur (genauer dazu in Abschnitt 5.3.4). Da das Produktionssystem plattformunabhängig Wrapper ausführen können soll, ist der *Internet Explorer* zusätzlich ungeeignet. Der *Mozilla*-Browser läuft zwar unter vielen Plattformen, besitzt jedoch keine einfache Schnittstelle zum Auslesen des Zerteilungsbaumes. Beide Browser sind relativ komplexe Systeme, so dass eine Nutzung als Basiskomponente mit Risiken bei der Implementierung verbunden ist. Es soll eine der in Abschnitt 3.2 beschriebenen Komponenten verwendet werden.

5.3 Benchmark für fehlertolerante PH-Zerteiler

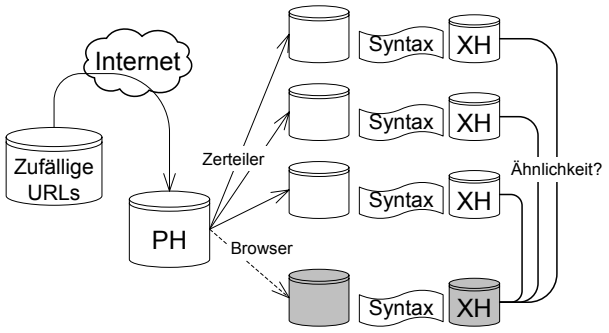


Abbildung 5.1: Schritte des Benchmarks

Die Kernidee des Benchmarks wird hier knapp beschrieben und anschließend Schritt für Schritt entwickelt. Die Abbildung 5.2 gibt einen Überblick über die Zusammenhänge. Es ist ein PH-Zerteiler auszuwählen, der ein XH-Dokument erzeugt, welches möglichst der vom Browser benutzen Baumdarstellung ähnelt. Diese Baumdarstellung ist durch JavaScript DOM-Methoden (ab level 2) zugänglich und eng mit der visuellen Darstellung gekoppelt. Diese Darstellung kann in XML-Syntax notiert werden und mit dem XH-Dokument über XML-Ähnlichkeitsmaße verglichen werden. Der Benchmark gliedert sich in folgende Teile (vergl. Abb. 5.1):

- Erzeugen einer *Testmenge von* PH*-Daten* mit Hilfe von zufälligen URLs
- *Aufbereiten von* PH *in* XH durch die zu prüfenden Zerteiler
- *Syntaktische Analyse* auf gültiges HTML bzw. wohlgeformtes XML
- *Möglichst semantischer Vergleich* der XH-Dokumente mit dem Zerteilungsergebnis eines Browsers in XH-Form (im der Grafik mit DOM bezeichnet). Um den Vergleich durchführen zu können, muss ein Browser-DOM per JavaScript in XH überführt werden. Die beiden XH-Dokumente können dann über ein noch zu definierendes XH-Ähnlichkeitsmaß verglichen werden.

[4] Schnittstelle speziell für *Windows*

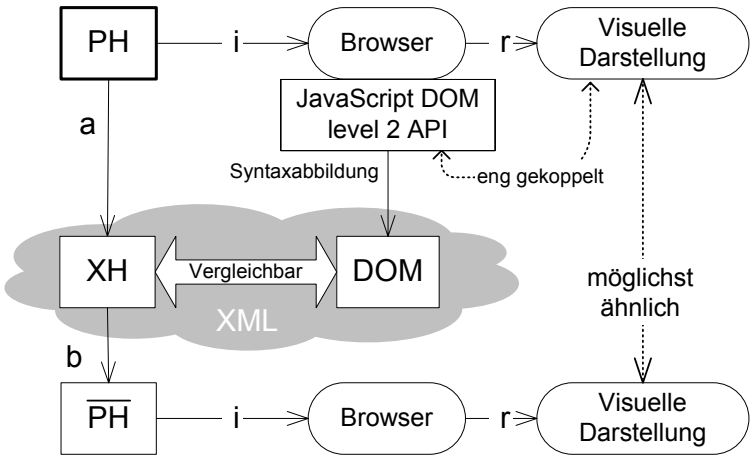


Abbildung 5.2: Kernidee des Benchmarks

5.3.1 Erzeugen einer Testmenge von PH-Daten

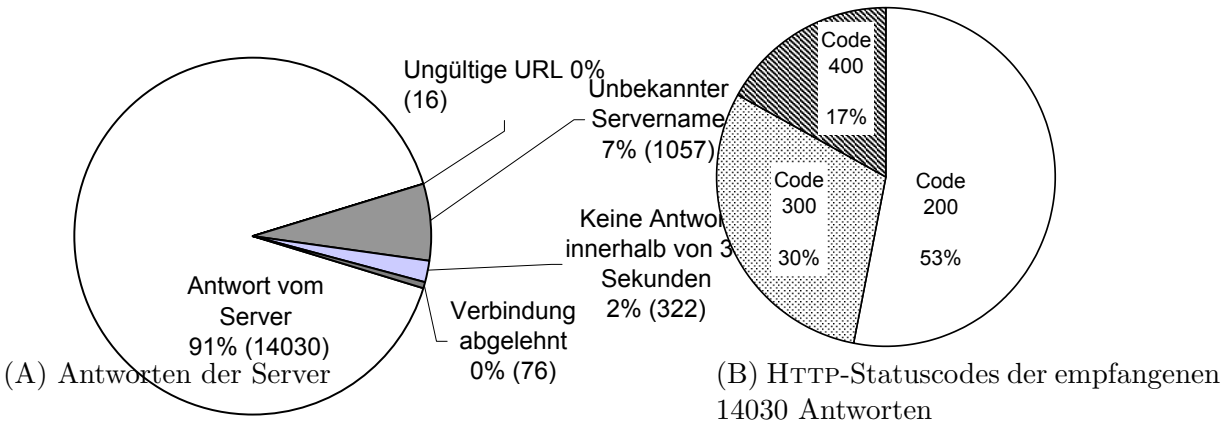
Um PH-Zerteiler überhaupt testen zu können, muss eine Testmenge von PH-Daten vorliegen, anhand der sie gemessen und verglichen werden können.

Für eine aussagekräftige Statistik wird eine gut gestreute Menge von zufälligen, bestehenden und realen Webseiten benötigt. Dazu muss eine Quelle für zufällige URLs genutzt werden, von denen es nur wenige gibt: Zufällige URLs

***Yahoo*[5]** liefert viele URLs, die auf nicht länger existente Webseiten verweisen. Die anderen zufälligen URLs verweisen vermutlich auch auf schon länger bestehende Webseiten. Von einer ersten Stichprobe mit 20.000[6] URLs ist z. B. fast jede zweite URL eine Doublette. Das lässt darauf schließen, das die URL-Datenbasis nicht besonders groß ist. Es lassen sich kurioserweise fast keine Informationen über die Herkunft der URLs aus dem *Yahoo Random Link Generator* finden, außer dass sie aus der Datenbasis der *Yahoo*-Suchmaschine kommen.

***Mangle*[7]** benutzt ein Wörterbuch, wählt daraus zufällig einige Begriffe und fragt *Google*[8] danach. Die URL des ersten Suchergebnisses ist die von *Mangle* erzeugte zufällige URL. Dieses Verfahren scheint geeignet, um tausende von aktuellen und gut gestreuten URLs zu erzeugen und sollte daher in Benchmarks bevorzugt verwendet werden.

[5] `http://random.yahoo.com/bin/ryl`
[6] Weitere Zahlen in dieser Arbeit stammen jeweils aus neuen Messungen, soweit nichts anderes angegeben ist.
[7] `http://www.mangle.ca`
[8] `http://www.google.com`



(A) Antworten der Server

(B) HTTP-Statuscodes der empfangenen 14030 Antworten
200 = OK, 300 = Weiterleitung, 400 = Fehler

Abbildung 5.3: Ergebnisse von HTTP-GET auf zufälligen URLs

Erzeugen der Testmenge

Aus einer Menge von eindeutigen, zufälligen `http://`-URLs werden die jeweiligen Zieldateien herunterladen und alle Daten der HTTP-Antworten für weitere Analysen gespeichert, also Header-Daten und der HTTP-Antwort-Nutzdatenteil, wobei die Kodierung vereinheitlicht wird. Etwa jede zehnte Anfrage (siehe Abb. 5.3 (A)) liefert keine HTTP-Antwort zurück, was fast immer an „Unbekannten Servernamen“ (Host not found) liegt. Für viele Anfragen antwortet der HTTP-Server nicht mit einer HTML-Seite, sondern mit einer Fehlermeldung (siehe Abb. 5.3 (B)). Als Testdaten werden nur die HTTP-Antwort-Nutzdaten verwendet, für die der Server mit „`200 OK`“ (normale HTTP-Antwort) geantwortet hat.

Anmerkung:
Dabei wurden HTTP-Antworten größer als 2 Megabyte manuell ignoriert (ca. 10). Zum Herunterladen der URLs wurde eine aktuelle Version des *Jakarta HttpClients* verwendet, der sich als „Internet Explorer“ identifizierte. Manche Server machen das gesendete PH von der Browser-Kennung in der Anfrage abhängig.

5.3.2 Aufbereiten von PH in XH

Die Testmenge der PH-Daten muss von den verschiedenen Zerteilern bearbeitet werden, um für jeden Zerteiler eine entsprechende Menge von XH-Dokumenten zu erhalten. Dazu müssen die bestehenden Zerteiler in Rahmen des Benchmarks verwendet werden.

Einbindung von PH-Zerteilern

Für die Einbindung bestehender Zerteiler wurde eine einheitliche Schnittstelle definiert: Eingabe ist eine PH-Zeichenkette aus welcher der Zerteiler ein W3C-DOM erzeugen und zurück liefern muss. Alternativ darf er genau *einen* vordefinierten Fehler melden, falls er die Eingabe nicht bearbeiten kann.

Anmerkung:
Für jeden Zerteiler muss zur Nutzung im Benchmark lediglich eine kleine Adapterklasse geschrieben werden, die von der vordefinierten abstrakten Klasse `Cleaner` erbt. Die Schnittstelle für den Benchmark ist identisch mit der Schnittstelle zum Modul `clean` und wird in Abschnitt 6.1.2 auf Seite 55 illustriert und erläutert.

Die Zerteiler liefern aus den PH-Seiten ein DOM, welches als XML (und damit XH) serialisiert wird. Die erzeugten XH-Dokumente können nun analysiert und mit anderen verglichen werden.

5.3.3 Syntaktische Analyse

Unter der syntaktischen Analyse wird hier die Prüfung auf gültiges HTML und wohlgeformtes XML verstanden. Zur Prüfung auf gültiges HTML wird der Online-Validator[9] des W3C benutzt, welches die Standards (X)HTML erstellt hat.

Seit der Erstellung der Testmenge können sich einige Webseiten verändert haben oder nicht mehr erreichbar sein. Um dies auszuschließen wurden die lokalen Testdaten auf einen öffentlich erreichbaren Test-Webserver gestellt und diese dem Online-Validator übergeben.

Validierung realer Webseiten

Zunächst wurde getestet, welcher Anteil der realen Webseiten bereits HTML-konform vorliegt. Von einer wie in Abschnitt 5.3.1 definierten Testmenge mit 5534 PH-Seiten waren es nur 13 Seiten, das entspricht lediglich 0,24 %. Damit hat der W3C-Test für PH-Seiten keine Aussagekraft.

Anmerkung:
Für den Test wurden zufällige URLS mit Hilfe von *Yahoo* erzeugt, da dem Autor zum Zeitpunkt der ersten Tests *Mangle* noch nicht bekannt war. Die mit Hilfe des *Yahoo Random Link Generators* erreichten Ergebnisse werden wegen der großen Datenbasis nicht von mit *Mangle* erreichbaren Ergebnissen abweichen.

Analyse von *JTidy*

JTidy[10] (beschrieben in 3.2) wurde mit einer Menge von 7476 PH-Dateien getestet. *JTidy* lieferte für manche Seiten gar kein Ergebnis, erzeugte aber in vielen Fällen zumindest wohlgeformtes XML (siehe Abb. 5.4). Gültiges (X)HTML im Sinne des W3C wurde bei jeder zehnten Ausgabe erzeugt. Da *JTidy* in so vielen Fällen nicht einmal *irgendeine* XML-Darstellung erzeugen konnte, muss ein besseres Werkzeug gefunden werden.

Erfahrungen mit Zerteiler-Komponenten

Die fehlertoleranten PH-Zerteiler-Komponenten haben im Test Probleme mit JavaScript-Anweisungen, da innerhalb eines solchen Blocks auch HTML-Syntax, z. B. als `document.write("<B>text</B>");` vorkommen darf. Des weiteren ist die korrekte Behandlung von *Whitespace*[11] schwierig – besonders für XML-nahe

[9] `http://validator.w3.org/`
[10] Version: 04aug2000r7-dev, 2001-08-01 08:19, unter Verwendung aller auf der Homepage zur Verfügung stehenden Patches
[11] Sammelbegriff für Leerzeichen, Tabulatoren und expliziten Zeilenumbrüchen

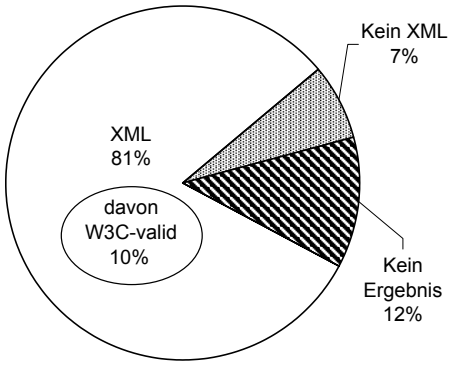


Abbildung 5.4: Ausgabe von *JTidy*

Zerteiler, da für Html und Xml unterschiedliche *Whitespace*-Konventionen gelten. Schließlich machen zahlreiche Webautoren und Webdesignwerkzeuge von der Möglichkeit Gebrauch, eigene Informationen in selbst definierte Elemente zu verpacken. Dies funktioniert, da Browser ihnen unbekannte Elemente ignorieren. Unbekannte Elemente, die nicht Teil des Html-Standards sind, sind für manche Zerteiler ein Problem, da hier einige Heuristiken zur Interpretation nicht anwendbar sind. Beispielsweise verwenden manche Zerteiler (z. B. *JTidy*) Regeln zur Schachtelung von Elementen, um zu entscheiden, welches von zwei miteinander verzahnten Elementen (`<B><I>...</B></I>`) das äußere werden soll. Für unbekannte Elemente gibt es im Zerteiler kein definiertes äußeres Element.

5.3.4 Möglichst semantischer Vergleich

Idealerweise werden die Zerteiler nicht nur syntaktisch analysiert, sondern semantisch mit ihrem Vorbild, einem Browser, verglichen. Der beste Zerteiler ist derjenige, der mit seiner Ausgabe dem Browser am nächsten kommt. Um diese Messung durchführen zu können, werden die Xh-Ausgabe eines Browsers sowie ein geeignetes Xh-Ähnlichkeitsmaß benötigt.

Ausgabe eines Browser-Zerteilers

Um die von einem Browser erzeugten Doms zu erhalten, gibt es folgende Möglichkeit: Ein Browser, der die Spezifikation von *JavaScript* Dom *level 2* implementiert – und das tun einige – stellt Operationen zum Auslesen des Dom-Baumes zur Verfügung (`getRootElement`, `getChildren`, ...). Damit ist das Dom prinzipiell über JavaScript auslesbar. Zum Auslesen der Dom-Bäume, die der Browser aus den Ph-Testdaten erzeugt, wird ein eigener Webserver aufgesetzt und das in Abb. 5.5 gezeigte Verfahren verwendet:

Auslesen des DOMs

Der Webserver liefert jeweils eine Seite aus der oben beschriebenen Testmenge mit einem angehängten JavaScript-Bereich am Ende einer Http-Antwort (1). Die Ph-Seite wird vom Browser als Baumstruktur aufbereitet (2) und das dort enthaltene Script wird nach dem vollständigen Laden der restlichen Seite ausge-

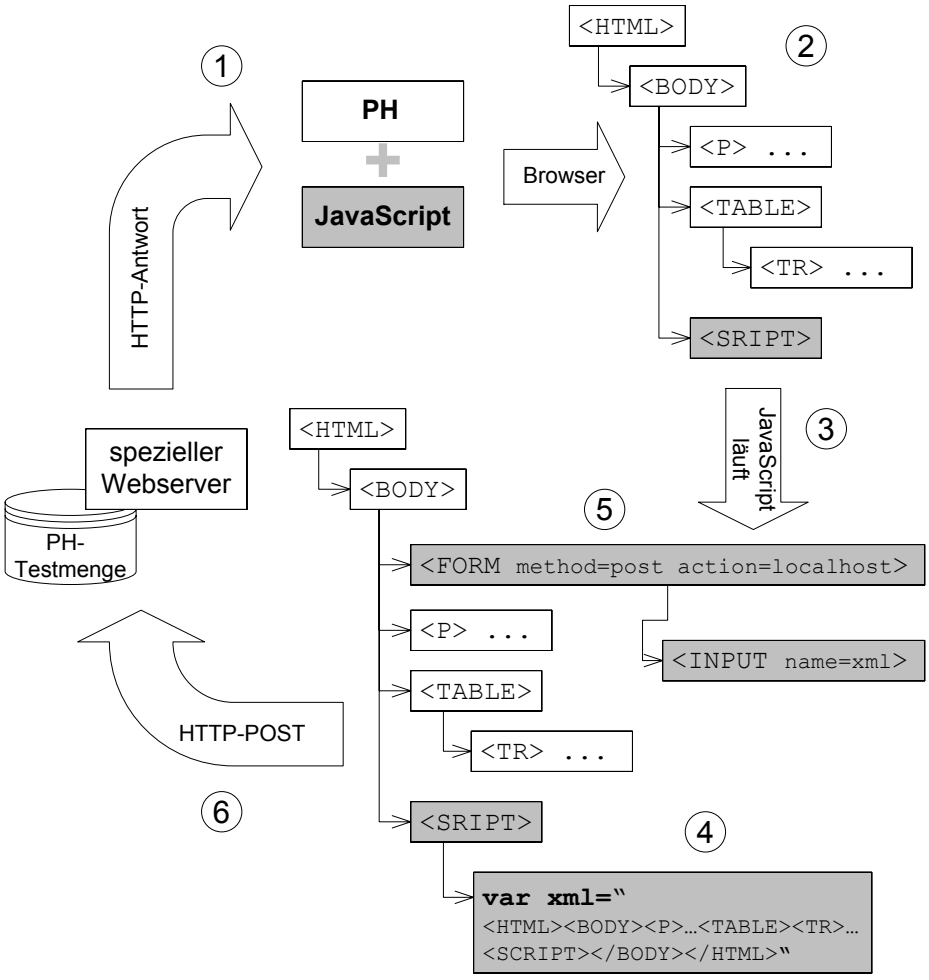


Abbildung 5.5: Browser liefert seine PH-Zerteilungsbäume

führt (3). JavaScript legt eine Variable für eine Zeichenkette (4) an und erzeugt rekursiv durch Ablaufen des HTML-DOMs im Browser eine XML-Repräsentation desselben. Dabei werden nur Elemente, Attribute und Kommentare betrachtet. Ebenfalls über die DOM-Befehle wird ein POST-Formular angelegt (5), mit dem die XML-Zeichenkette an den Server gesendet wird (6). Der spezielle Server liefert als Antwort daraufhin eine weitere PH-Seite mit angehängtem JavaScript zurück (1). So können automatisch viele vom Browser erzeugte DOMs als XH gewonnen werden.

Internet Explorer nicht verwendbar

Naheliegend wäre es, den marktbeherrschenden Browser *Internet Explorer* zu verwenden. Dieser gibt in Tests gegenüber der JavaScript-Schnittstelle zwar sein DOM preis, es ist aber nicht das gleiche, welches im Browser visuell dargestellt

wird. Ein Beispiel ist in Abb. 5.6 abgebildet. Bei Nutzung der COM-Schnittstelle[12] wurde dabei das gleiche Verhalten wie bei JavaScript beobachtet. Es wurde der gleiche (falsche) Baum wie über JavaScript zurückgeliefert. Daher wurde im weiteren Vorgehen *Opera* (7.01) verwendet.

Anmerkung:
Um die Abarbeitung der JavaScript-Programme zu beschleunigen, wurde ein Rechner als Server und ein anderer als Client bestimmt. Auf diesem liefen in mehreren Fenstern gleichzeitig die DOM-Sende-Programme, wobei die Bilder abgeschaltet wurden. *Opera* blieb bei ca. jeder zehnten Seite hängen und die Seite musste manuell neu geladen werden, worauf der Server eine neue Seite lieferte – für die ursprüngliche Seite kann *Opera* kein XH erzeugen. Bei ca. jeder hundertsten Seite musste der ganze Browser neu gestartet werden. Insgesamt konnten so in einigen Stunden mehrere tausend DOMs überwacht gewonnen werden.

- PH-Quelle:
```
<p><b>aaa<p>bbb</b>ccc<SCRIPT> ... </SCRIPT>
```

- Darstellung über JavaScript:
```
<HTML>
  <HEAD><TITLE></TITLE></HEAD>
    <BODY>
      <P><B>aaa<P>bbbccc<SCRIPT>...</SCRIPT></P></B></P>
      <P>bbbccc<SCRIPT>...</SCRIPT></P>
    </BODY>
</HTML>
```

- Visuelle Darstellung im Browser:
  aaa

  bbbccc

Abbildung 5.6: Unterschiede zwischen visueller Darstellung und JavaScript-Schnittstelle beim Internet Explorer
In der JavaScript-Darstellung kommen Elemente doppelt vor (bbbccc)

[12] Basierend auf dem Beispielcode von *Nicolas LeBlanc* zu seinem Artikel „Parsing HTML without Using the Browser“, `http://www.codeguru.com/vb_internet/htmlparser.html`

XH-Ähnlichkeitsmaß

Um die Ähnlichkeit der XH-Dokumente aus Zerteiler-Komponenten mit denen aus einem Browser zu messen, ist ein rein syntaktischer Vergleich wenig sinnvoll, da z. B. für das PH-Fragment

```
<B><I>Text</B></I>
```

unter anderem die zwei verschieden Darstellungen

```
<B><I>Text</I></B>
```
(a)

und

```
<I><B>Text</B></I>
```
(b)

in XH existieren, denn beide Varianten (a) und (b) werden in einem Browser gleich dargestellt. Syntaktisch ist die Gleichheit hier aber nicht erkennbar. Die erzeugten XH-Bäume müssen aus diesem Grund möglichst semantisch miteinander verglichen werden.

Die Position im XH-Baum spielt für die Darstellung eine noch wichtigere Rolle, als die auf ein Text-Element wirkenden Formatierungen, also alle Vater-Elemente und deren Attribute. Daher ist die Position eines Elementes beim semantischen Vergleich besonders zu betrachten. Ein wirklich semantischer Vergleich kann automatisch nicht einfach durchgeführt werden. Im folgenden wird ein handhabbares Ähnlichkeitsmaß für XH-Dokumente entwickelt.

Editierdistanz auf Bäumen als Grundlage für ein Ähnlichkeitsmaß

Eine nahe liegendes Ähnlichkeitsmaß für XH-Bäume ist die *minimale Editierdistanz auf Bäumen*, für die es einige unterschiedliche Definitionen gibt [Mou02], je nach erlaubten Basis-Operationen im Baum. Mögliche Operationen sind beispielsweise das Einfügen, Löschen oder Verschieben eines Teilbaumes oder einzelner Elemente.

Wünschenswert ist eine Editierdistanz, die das Kopieren und Einfügen von ganzen Teilbäumen berücksichtigt. Dieses Problem ist NP-vollständig [Mou02] und zudem ist keine frei verfügbare Implementierung bekannt. Die Entwicklung eines Werkzeugs zur Berechnung der minimalen Editierdistanz auf XML-Bäumen ist im Rahmen dieser Arbeit zu aufwendig.

Auch für Heuristiken zur Bestimmung einer – nicht notwendigerweise minimalen – Editierdistanz existiert Literatur [Lin01] (ausführlich) und [NJ02] und auch einige Implementierungen sind verfügbar. Allerdings brechen die meisten Implementierungen den Vergleich von zwei syntaktisch wohlgeformten XML-Dateien mit Fehlermeldungen ab, die darauf hindeuten, dass die Dateien zumindest korrekt eingelesen wurden.

- *treediff* (Version 1.1, 2000) von *Christian Nentwich* berechnet keine minimale Editierdistanz und behandelt nur folgende Baumoperationen: Einfügen von neuen Elementen in eine Elementliste, Löschen eines Teilbaumes sowie Veränderungen eines Attributes oder Textknotens.
- Das *XML Diff and Merge Tool* (2001) von *IBM Alphaworks* baut auf *XML4J* auf und bricht beim Aufruf mit einer Fehlermeldung ab.
- *xmldiff* (Version 0.6.3, 2001) von *Sylvain Thenault, Logilab* ist in Python implementiert und bricht ebenfalls mit einer Fehlermeldung ab.

- *3DM merging and differencing tool for* XML (Version 1.5, 2001) von *Tancred Lindholm* beherrscht das Einfügen und Löschen von Teilbäumen. Aufgrund von Heuristiken liefert dieses Werkzeug keine minimale Editierdistanz, läuft dafür aber so schnell, dass es im Rahmen dieser Arbeit verwendet werden kann. In der dazugehörigen Arbeit [Lin01] wird der Aufwand als proportional zur Größe der Eingabebäume angegeben.

  Anmerkung:
  In der Praxis läuft das Werkzeug bis zu einer Eingabegröße von ca. 250 KB sehr schnell und wird bei größeren Dateien merklich langsamer.

- *HTMLDiff* (Version 1.2, 2002) von *ComponentSoftware* liefert nur über eine manuell zu bedienende grafische Schnittstelle einen Vergleich von zwei XML oder HTML-Dokumenten. Daher kann dieses Werkzeug nicht automatisiert im Rahmen des Benchmakrs genutzt werden. Zusätzlich wird fälschlicherweise die Reihenfolge der Attribute berücksichtigt, welche in XML und HTML keine Rolle spielt.
- *diffmk* (Version 2.0.b1, 2002) von *Norman Walsh* bricht stets mit einer Fehlermeldung ab.
- *vmtools* (Version 0.5, 2002) von *VM Systems* beherrscht Einfügen und Löschen von Teilbäumen und bricht mit einer `NullPointerException` ab.
- *XmlDiffPatch* (Version 1.0, 2002) von *Microsoft* ist eine reine Bibliothek für das *.NET*-Rahmenwerk und kann daher nicht ohne weiteres verwendet werden.
- *diffxml* [Mou02] (2003) von *Adrian Mouat* bricht bei vielen XML-Dokumenten ab.

Mit 3DM steht nur ein Baumvergleicher zur Verfügung, der keine *minimale* Editierdistanz liefert, sondern aufgrund von Heuristiken letztlich eine "verrauschte" Distanz liefert. Das heisst, es wird angenommen, dass die heuristische Editierdistanz mit der minimalen Editierdistanz korreliert. Es bleibt experimentell (mit einem Werkzeug zur Berechnung der minimalen Editierdistanz auf Bäumen) zu ermitteln, ob die gewonnenen Abstandsinformationen mit dem minimalen Abstand hinreichend stark korrelieren, um dieses Maß zu rechtfertigen.

Mit Hilfe der minimalen Editierdistanz von zwei XML-Dokumenten kann ein Ähnlichkeitsmaß definiert werden. Die Ähnlichkeit $\overline{AB_{3DM}}$ von zwei XH-Bäumen A und B bezüglich einer Menge von XML-Bäumen M wird mit Hilfe von 3DM definiert als:

$$\overline{AB_{3DM}} = 1 - \frac{Abstand(A,B) - Min(M)}{Max(M)} \tag{5.1}$$

mit Min(M) als dem Minimum der paarweisen Abstände von Bäumen ($Abstand(A,B)$) in M (Max(M) analog), um stets einen Wert zwischen 0 und 1 zu erhalten.

Ein neuer Baumvergleich für XH

Aufgrund der Schwierigkeiten bei vorhandenen Abstandsmaßen wurde ein neues Maß entwickelt. Ein Maß, welches aus der Häufigkeit der vorkommenden Elemente, unabhängig von ihrer Position, berechnet wird, hat in der Praxis eine geringe Unterscheidungskraft [NJ02]. Also muss die Position eines Elementes berücksichtigt werden. Da sich vermutlich Positions-Fehler näher an der Wurzel des XH-Dokumentes stärker auf das visuelle Erscheinungsbild auswirken, wird die Ähnlichkeit von zwei Bäumen über das Verhältnis ihres *größten gemeinsamenen Schnittbaumes* (ggS) zu ihrer eigenen Größe definiert.

Größter gemeinsamer Schnittbaum (ggS)

Der *größte gemeinsame Schnittbaum* von zwei Bäumen A und B sei definiert durch einen Algorithmus, der auf beiden zu vergleichenden XH-Bäumen parallel eine Breitensuche durchführt, solange die Elemente paarweise gleich sind.

Anmerkung:
Eine Erweiterung auf n Bäume ist möglich, wenn der aus zwei Bäumen errechnete Schnittbaum wieder als XML-Dokument dargestellt wird und die ursprünglichen Bäume der Menge ersetzt.

Für zwei Bäume A und B wird der Schnittbaum rekursiv berechnet. Im ersten Schritt werden die beiden Wurzelknoten verglichen. Wenn sie sich *ähnlich* sind, wird eine der beiden Wurzelknoten der Wurzelknoten des gemeinsamen Schnittbaumes, andernfalls ist der Schnittbaum leer.

Nun werden die Kinderelemente der Knoten a (Wurzeln von A) und b (Wurzeln von B) verglichen, dabei werden nur Elemente und Textknoten betrachtet. Die beiden Kinderlisten werden verglichen, *ähnliche* Knoten werden wieder in den Schnittbaum aufgenommen, und rekursiv *deren* Kinder verglichen.

Ähnliche Knoten

Reine Textknoten werden über den Inhalt verglichen. Zwei Elementknoten gelten als *ähnlich*, wenn sie den gleichen Namen besitzen, wobei die Groß-/Kleinschreibung vernachlässigt wird, da sie in PH für das visuelle Erscheinungsbild keine Bedeutung hat. Die Attribute von Elementen können ignoriert werden (Variante **-A**), da Attribute im Browser nicht dargestellt werden, sondern lediglich Informationen enthalten, *wie* der Inhalt der Elemente dargestellt werden soll. Alternativ kann auch die Gleichheit von Namens-/Wertpaaren gefordert werden (Variante **+A**). Es muss empirisch untersucht werden, in wie weit die beiden Varianten unterschiedliche Ergebnisse liefern.

Ähnliche Unterelementlisten

Zum Vergleich von zwei Listen von Unterelementen gibt es zwei unterschiedliche Varianten. Die einfachere (mit **strict** bezeichnet) fordert, dass sich die beiden Listen in Länge gleichen und zudem für jede Position in der Liste die jeweiligen Elemente *ähnlich* sind. In der Variante **med** wird für die beiden Listen die minimale Editierdistanz für Zeichenketten [Lev66] verwendet. Die Elementlisten werden dazu als Zeichenketten aufgefasst und umgeordnet, um maximal viele Elemente, die sich *ähnlich* sind, aufeinander abzubilden.

Anmerkung:
Die Kosten für das Ändern eines Knotens sind größer, als für das Löschen und Einfügen zusammen, um zu verhindern, dass der Algorithmus Elemente lediglich gegeneinander austauscht statt mit Löschen und Einfügen die Struktur des Baumes zu berücksichtigen.

Ergebnis

Wenn die Breitensuche nicht weiter absteigt, ist der *ggS* berechnet. Alle Elemente des *ggS* entsprechen damit XH-Elementen, die in beiden Bäumen an gleicher Position (u.U. abgesehen von der exakten Reihenfolge) als Kinder-Elemente vorkommen.

Definition eines Abstandsmaßes mit Hilfe des *ggS*

Die Anzahl der Elemente im *ggS* sei c.
Der erste Baum des Vergleichs heiße A, die Anzahl seiner Elemente sei a,
der zweite Baum sei B, seine Element-Anzahl b.

Dann ist die Ähnlichkeit $\overline{AB_{ggS}}$ von A und B definiert als

$$\overline{AB_{ggS}} = \frac{c}{Max(a,b)} \tag{5.2}$$

Für $A = B$ ist $C = A(= B)$ und damit $c = a(= b) \Rightarrow \overline{AB} = 1$. Für $c = 0$, also bereits verschiedene Wurzel-Elemente gilt $\overline{AB} = 0$.
Damit gilt: $0 \leq \overline{AB} \leq 1$.

5.3.5 Ergebnisse des Benchmarks

Wohlgeformtes XML?

Auf die Testmenge von PH-Zeichenketten werden verschiedene Zerteiler angewandt und ihre Ergebnisse als XH gespeichert. Manche Zerteiler liefern teilweise keine Ausgabe. Anschließend prüft ein Testlauf, ob wohlgeformte XML-Dokumente erzeugt wurden (vergl. Abb. 5.7).

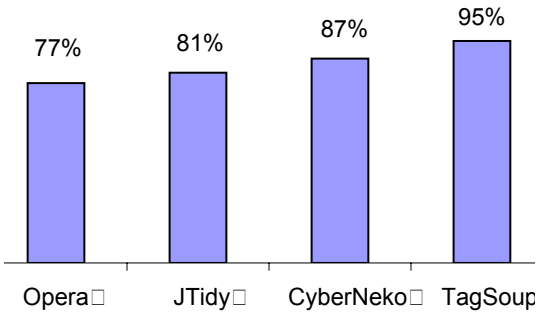
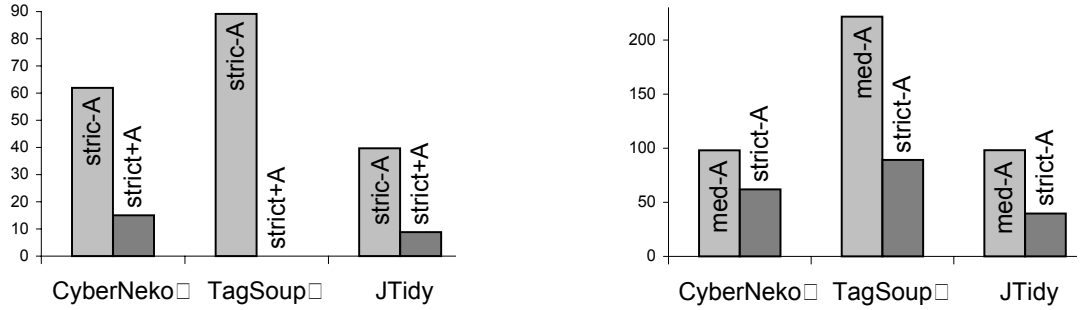


Abbildung 5.7: Ausgabe von PH-Zerteilen (inkl. *Opera*)
Anteile der PH-Daten, für die wohlgeformtes XML erzeugt wurde. Angaben in Prozent.

Vergleich mit Browser-XH

Die korrekten XH-Dokumente werden mit dem Browser-XH verglichen. Wenn für die Gleichheit von zwei Elementen auch die Gleichheit der Attributnamen und -werte eines Elementes gefordert wird (Variante **+A**), verändert sich das Ergebnis drastisch (siehe Abb. 5.8 A).

Während die Variante **-A** als Favorit *TagSoup* ermittelt, wertet **+A** diesen als Schlusslicht. Der Grund ist vor allem das von *TagSoup* stets eingefügte XML-Namespace-Attribut, welches von *Opera* und den anderen Werkzeugen nicht erzeugt wird. Ein Beispiel für eine PH-Datei und die daraus erzeugten XH-Bäume



(A) Variante **strict-A** und **strict+A** (B) Variante **med-A** und **strict-A**

Abbildung 5.8: Varianten von LCT zur Messung der Ähnlichkeit zu *Opera*
Y-Achsen: Summe der $\overline{AB}$-Werte

der PH-Zerteiler findet sich auf Seite 90ff. Es stellt sich die Frage, ob die strikte Forderung nach Ähnlichkeit aller Kinderelemente (Variante **strict**) zu streng ist. Ein Vergleich mit der Variante **med** zeigt aber, dass die größten gemeinsamen Schnittbäume absolut zwar größer werden (und damit die errechneten Ähnlichkeiten), sich das relative Verhältnis der Ähnlichkeiten zueinander dadurch aber nicht verändert (Abb. 5.8 B). Aus diesem Grund wird im folgenden die Variante **strict-A** als Grundlage des Benchmarks verwendet und mit LCT (largest common tree) bezeichnet.

Es bleibt noch zu prüfen, ob die Länge des von 3DM gelieferten Editierskriptes und die daraus berechnete Ähnlichkeit mit denen von LCT korreliert. Die absoluten Ähnlichkeiten von 3DM liegen höher als die von LCT-*strict-A* (Abb. 5.9) und liefern auch eine andere Reihenfolge. Die relativen Ähnlichkeiten zeigen allerdings, dass 3DM kaum Unterschiede zwischen den verschiedenen PH-Zerteilern misst, und die gelieferte Reihenfolge damit weniger aussagekräftig ist, als die von LCT (Abb. 5.10).

Bei beiden Ähnlichkeitsmaßen ist *TagSoup* am ähnlichsten zum Browser *Opera* und wird daher in d_2c verwendet.

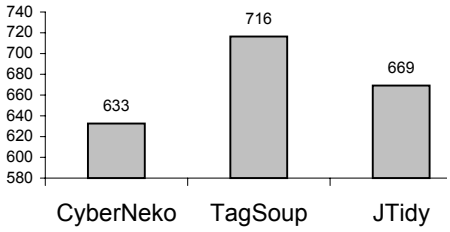


Abbildung 5.9: Absolute $\overline{AB}$-Werte von 3DM
Y-Achse: Absolute Anzahl von Anweisungen des Editierskriptes

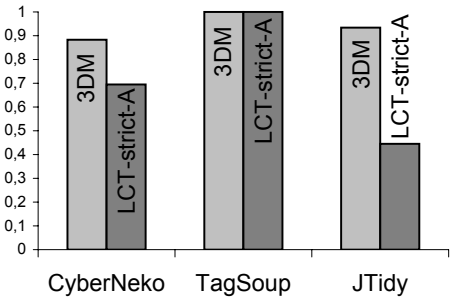


Abbildung 5.10: Ähnlichkeiten von 3DM im Vergleich zu LCT
Größter Wert jeweils normiert auf 1

5.4 Auswahl einer Komponente des Moduls proxy

Da der Proxy nur aus einem schlanken, anwendungsspezifischen Kern besteht, der sich zwischen einem *Servlet*-fähigen Webserver und dem Modul *surf* befindet, ist ein *Servlet*-fähiger Webserver auszuwählen. Dieser sollte einfach zu konfigurieren sein und bestehende Standards beherrschen.

Folgende *Servlet*-fähige Server wurden betrachtet:

Tiny Java Web Server ist sehr „schlank“, bietet aber nur recht umständliche Konfigurationsmöglichkeiten. Da während der Umsetzung verschiedene Konzepte getestet werden sollen, kommt dieser Server für die Umsetzung nicht in Frage. Er bietet nur rudimentäre Servlet-Unterstützung.

Jetty 4.1.4 stable wird in Entwicklerkreisen als sehr flexibel gelobt und ist einfach über eine Java-Programmier-Schnittstelle zur Laufzeit zu konfigurieren.

Tomcat 4.1.24 ist als JSP-Referenzserver etwas zu „mächtig“ für einen in ein System eingebetteten Server, der zudem JSP nicht nutzt. Hier wird die Konfiguration zwar nochmals flexibler, jedoch damit auch deutlich umständlicher.

Ausgewählt wurde *Jetty*, da es sowohl HTTP/1.0 als auch HTTP/1.1 unterstützt, HTTPS-fähig ist, einfach zu konfigurieren ist, eine übersichtliche Programmier-Schnittstelle bietet, aber über die *Jasper*-Engine optional auch JSP-Seiten verarbeiten kann. Diese Option wird dann interessant, wenn die BenutzerSchnittstelle außerhalb der eigentlichen Auftragserstellung komplexer wird.

5.5 Auswahl eines Dom-Paketes

Ein Dokumentmodell muss einerseits die W3C-DOM-Spezifikation für XML einhalten, andererseits dem Entwickler möglichst kompakte Operationen erlauben. Die Auswahl eines Dokumentmodells beeinflusst stark den Speicheraufwand des Gesamtsystems.

Ein austauschbares Dokumentmodell erfordert häufige Konvertierungen von einem Modell in ein anderes, was durch vollständiges Ablaufen des Quellmodells erfolgt, während ein ganz neues Dokument im Zielmodell erstellt wird. Dieser Prozess ist langsam und aufgrund der Komplexität von DOM nicht einfach. Daher soll auf ein austauschbares DOM verzichtet werden. Änderungen am DOM ziehen dann unter Umständen Änderungen am DOM-erzeugenden Modul *clean* und am DOM-transformierenden Modul *extract* nach sich. Ebenso sind möglicherweise die Module *ui* und *learn* anzupassen. Eine Konvertierung zur Laufzeit ist so nur noch einmal vom Modul *clean* zum Modul *extract* nötig.

Java-DOM ist das direkt aus der W3C-Spezifikation abgeleitete DOM. Es besitzt kaum Funktionalität über die reinen DOM-Funktionen hinaus und nutzt daher die Sprache Java sehr unzureichend. Operationen im Java-DOM sind teilweise umständlich oder sogar unmöglich.

JDom Beta 8 war das erste Java-zentrierte DOM. Es nutzt konkrete Java-Klassen, wodurch eine einfache Programmier-Schnittstelle mit geringer Flexibilität realisiert wird. *JDom* kann Java-DOM als Eingabe verarbeiten und in der neusten Version ist nun sogar XPath-Unterstützung enthalten. Das Paket wird als *open source* unter einer Apache-ähnlichen Lizenz gepflegt und wird mittlerweile als *Java Specification Request* (JSR-102) innerhalb des *Java Community Process* (JCP) standardisiert.

Dom4j 1.3 ist ursprünglich als Abkömmling des *JDom*-Projektes entstanden. Es verwendet Interfaces statt Klassen, bietet XPath-Unterstützung und zusätzlich zu einer eigenen, reichhaltigen Programmier-Schnittstelle mit JAXP[13]/TrAX[14]-Schnittstelle für die in d_2c wichtigen XSL-Transformationen, weiterhin die Java-DOM-Schnittstellen für größere Kompatibilität. *Dom4j* stützt sich stark auf den *Collection*-Klassen von Java und ist insgesamt wesentlich flexibler und weniger strikt als *JDom*. Es wird unter einer *BSD*[15]-ähnlichen Lizenz gepflegt.

XOM ist eine neues XML-Objekt-Modell. Es bietet baumbasierte Programmierschnittstellen für XML-Bearbeitung in Java und legt großen Wert auf korrektes XML und eine schlanke Programmier-Schnittstelle. Es wird von einer einzigen Person unter der *GNU Lesser General Public License* (LGPL) entwickelt.

Das System verwendet das mächtige *Dom4j*-Paket (`Dom4j-patched-1.3`) für die interne DOM-Repräsentation, da es die meiste Funktionalität besitzt und von einer Entwicklergemeinde aktiv weiter entwickelt wird.

[13] Java API for XML Processing
[14] Transformation API for XML
[15] Berkeley Software Distribution

6 Umsetzung/Implementierung

Wie im Kapitel Entwurf dargelegt, wird d_2c in zwei Schichten realisiert. Die höhere Schicht, das Entwicklungssystem, nutzt die untere Schicht, das Produktionssystem. Das komplette System ist in Java implementiert.

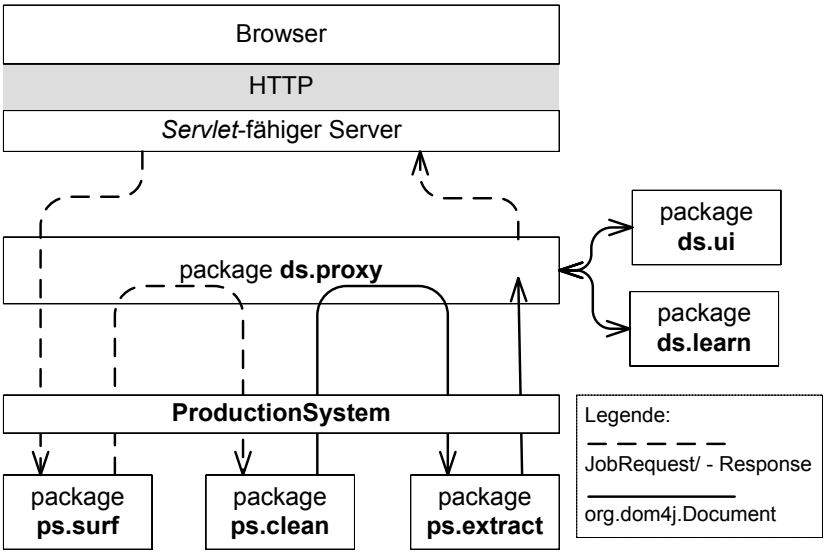


Abbildung 6.1: Kommunikation der beiden Schichten und Zusammenspiel der Module

Abb. 6.1 zeigt grob das Zusammenspiel der verschiedenen Module, die jeweils als eigenes Paket (`package`) in Java implementiert wurden. Ein *Servlet*-fähiger Server nimmt vom Browser Anfragen entgegen, wandelt sie in einen `JobRequest` um und schickt diesen an den *proxy*. Dieser nutzt das Produktionssystem, um den `JobRequest` tatsächlich als HTTP-Anfrage auszuführen.

Soll eine Seite instrumentalisiert werden, so lässt das Modul *proxy* die erhaltene `JobResponse` mit dem PH-Text vom Modul *clean* in ein XH-Dokument aufbereiten. Im restlichen Teil des Systems kommunizieren die Module dann nur noch über die XH-Dokumente. Erst auf dem Weg zum Browser zurück werden sie im Modul *proxy* wieder in PH serialisiert und in eine `JobReponse` verpackt. Der *Servlet*-fähige Server sendet diese dann als HTTP-Antwort zurück.

Die Abbildung macht die Rolle des Produktionssystems als Fassade vor seinen drei Untermodulen deutlich. Zu sehen ist auch, dass nur zwei interne Datenstrukturen zur Kommunikation zwischen den Schichten verwendet werden:

`JobRequest`/`-Response` zur Kapselung von HTTP-Nachrichten und `org.dom4j.Document` als Dokumentmodell. Das Modul *clean* ist in seiner Roller für die Konvertierung zu erkennen.

6.1 Produktionssystem

Das Produktionssystem sendet HTTP-Anfragen ab (*surf*), bereitet den empfangenen PH-Text zu XH auf (*clean*) und extrahiert mit einem XSLT-*Stylesheet* die gewünschten Daten (*extract*).

Wie in Abb. 6.2 illustriert, nutzt das Produktionssystem dazu Referenzen auf die Objekte in `ps.surf` und `ps.clean`. Die Extraktion wird durch Aufrufe an das statische Modul `extract` delegiert.

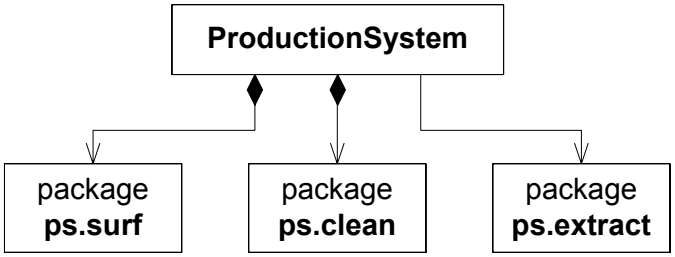


Abbildung 6.2: Paket-Diagramm des Produktionssystems

6.1.1 Modul surf

Dieses Modul soll genau wie ein Browser mit beliebigen Webservern kommunizieren. Dazu muss es z. B. eine frei definierbare Browser-Kennung senden und mit *Cookies* umgehen können.

Da zu Beginn der Arbeit nicht entschieden werden konnte, welche Implementierung die geeignetste sein würde und zudem in Zukunft neue, bessere Implementierungen verfügbar sein werden, muss die Implementierung vom Rest des System gut entkoppelt werden. Abb. 6.3 zeigt ein geeignetes Klassendiagramm: Nach außen sichtbar sind nur eine statische Fabrik-Klasse, eine Schnittstelle für den HTTP-Client (`Surfer`), sowie die Klasse einer möglichen Fehlermeldung (`UnableToSurfException`). Die Schnittstelle (`interface Surfer`) gibt nur eine Methode vor, die von den internen Datenstrukturen `JobRequest` und `JobResponse` Gebrauch macht.

```
public JobResponse getResponse( JobRequest jobRequest )
    throws UnableToSurfException;
```

Die beiden Job-Klassen abstrahieren anwendungsspezifisch von den HTTP-Anfragen und -Antworten, indem sie eine Reihe von zusätzlichen Statusfeldern und

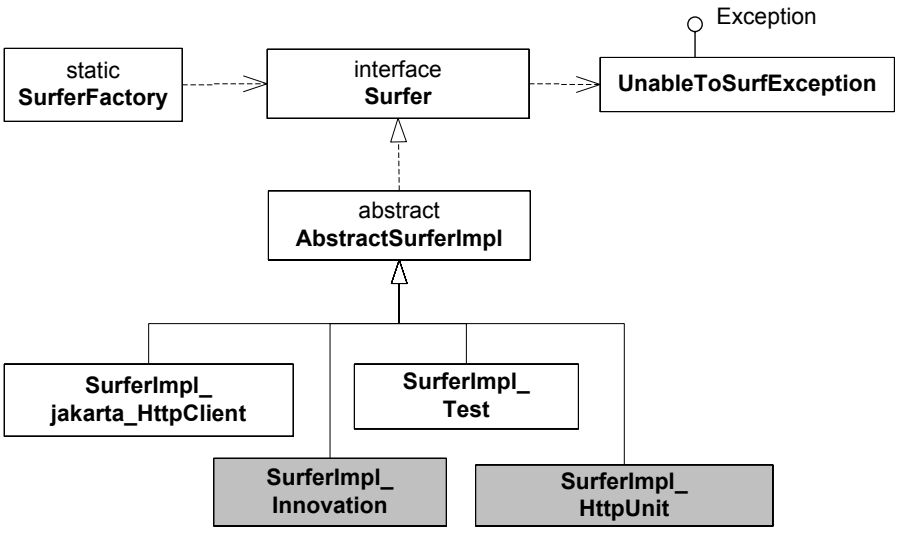


Abbildung 6.3: Klassendiagramm des Moduls *surf*

Funktionen einführen. Eine genauere Beschreibung dieser systemweit genutzten Klassen (`ps.common`) findet sich in 6.3.

Spezifische Implementierungen

Für jede Implementierung wurde eine Adapterklasse geschrieben, welche die `JobRequest` und `-Reponse`-Datenstrukturen in die spezifischen Anfragen der HTTP-Client-Implementierung konvertiert.

Die verwendete Implementierung *Jakarta Commons HttpClient* konnte in der Entwicklungsphase mit allen Webservern problemlos kommunizieren. Einzig der Webserver des Online-Auktionshauses *Ebay*[1] bereitete anfangs Schwierigkeiten, da er nach einem *Login* in den HTTP-Antwort-Headern erneut mehrmals HTTP-Statuszeilen sendet. Diese versucht der HttpClient beim Zerteilen als Header zu interpretieren und meldet einen Fehler. Der Quelltext des HttpClients wurde geändert, um falsch formatierte Header zu ignorieren. Hier hat sich die Strategie, auf *open source*-Komponenten zu setzen, bewährt. Eine reine Testimplementierung erzeugt aus der HTTP-Anfrage eine übersichtliche HTML-Seite, in der alle Daten der Anfrage zusammengestellt sind. Dadurch konnten Fehler schneller gefunden werden. Zeitweise wurden zwei weitere Implementierungen `SurferImpl_Innovation` und `SurferImpl_HttpUnit` verwendet. Beide waren aber nicht vollständig und flexibel genug, um in *surf* genutzt zu werden.

Filtern des Http-Datenstroms

Die Klasse `AbstractSurferImpl` manipuliert eingehende HTTP-Anfragen (`JobRequest`) und löst so eine Reihe von Problemen, unabhängig von der verwendeten Implementierung des HTTP-Clients:

Kodierung

Einige Server senden komprimierte Antworten (gzip). Um kein Dekomprimierungsmodul einbinden zu müssen, wird der Server aufgefordert, nur „plain text"-Antworten zu senden. Dazu wird der Header „`Accept-Encoding`" auf „`identity, *;q=0`" gesetzt.

[1] `http://www.ebay.de`

Undokumentierte Proxy-Standards

Der Header „`Proxy-Connection`“ muss von korrekten Proxys stets entfernt werden. Da das Modul *surf* auch Teil des Proxys ist, wird dies hier erledigt.

Anmerkung:
In einer Mailingliste schreibt *Marc Slemko*, Mitglied der *Apache Software Foundation*: All proxies should always strip the Proxy-Connection header from requests that are forwarded, ie. it is a hop-by-hop header. This is one of the several undocumented HTTP „must do's“ that software needs to deal with.

Browser-Identifikation

Da einige Webserver abhängig von der Identifikation des Browsers verschiedene Webseiten liefern, muss hier exakt die gleiche Identifikation durch das Modul `surf` gesendet werden. Dazu wird der Header „`User-Agent`“ auf die in der Konfigurationsdatei angegebene Zeichenkette gesetzt – Standardwert ist hier der *Internet Explorer 6.0*.

Zwischenspeicherung (Caching)

Webserver können auf eine HTTP-Anfrage des Browsers sinngemäß antworten: „Diese Antwort wurde bereits 1:1 überliefert und wird daher nicht noch einmal übertragen“ (Code 304). Dadurch kann es passieren, dass der Nutzer Daten im Browser sieht, die er nicht über d_2c geladen hat. Sie werden aus dem Browser-internen Zwischenspeicher geliefert. Für diese Seiten kann d_2c keine Extraktionsoberfläche darstellen. Daher werden die Anfragen des Browser vor dem Absenden so modifiziert, dass der Webserver nicht herausfinden kann, ob der Browser die Anfrage schon einmal so gestellt hat. Die Header „`If-Modified-Since`“ und „`If-None-Match`“ werden dazu entfernt.

6.1.2 Modul clean

Dieses Modul entscheided maßgeblich darüber, ob d_2c sich – wie gefordert – einem Browser gleich verhält oder nicht. Zur Entkoppelung von der eigentlichen Implementierung wurde hier der gleiche Ansatz wie im Modul *surf* gewählt (siehe auch Abb. 6.4): Eine statische Fabrik-Klasse (`CleanerFactory`) liefert eine Implementierung, die von der abstrakten Klasse `Cleaner` erbt. Diese Klasse ist auch Schnittstelle des auf Seite 38ff entwickelten Benchmarks. Sie ruft die unterliegende Implementierung auf, prüft das Ergebnis, wandelt von der `org.w3c.dom`-Implementierung in die im Rest des Projektes gebräuchliche `org.dom4j`-Implementierung und führt eine Nachbearbeitung des erhaltenen XH-Baumes durch. Auftretende Fehler werden nur durch eine einzige Fehler-Klasse (`UnableToClean Exception`) gemeldet, was zur weiteren Entkoppelung von Implementierung und Gesamtsystem beiträgt.

Aus drei Implementierungen von `Clean` kann über die Konfigurationsdatei ausgewählt werden: *JTidy*, *CyberNeko* und *TagSoup* stehen dem Anwender zur Verfügung. Weitere Implementierungen können leicht eingebunden werden, da nur eine Methode innerhalb einer Adapterklasse implementiert werden muss, die aus einer Zeichenkette einen XH-Baum erzeugt:

```
protected abstract org.w3c.dom.Document cleanImpl(String httpResponseBody)
    throws UnableToCleanException;
```

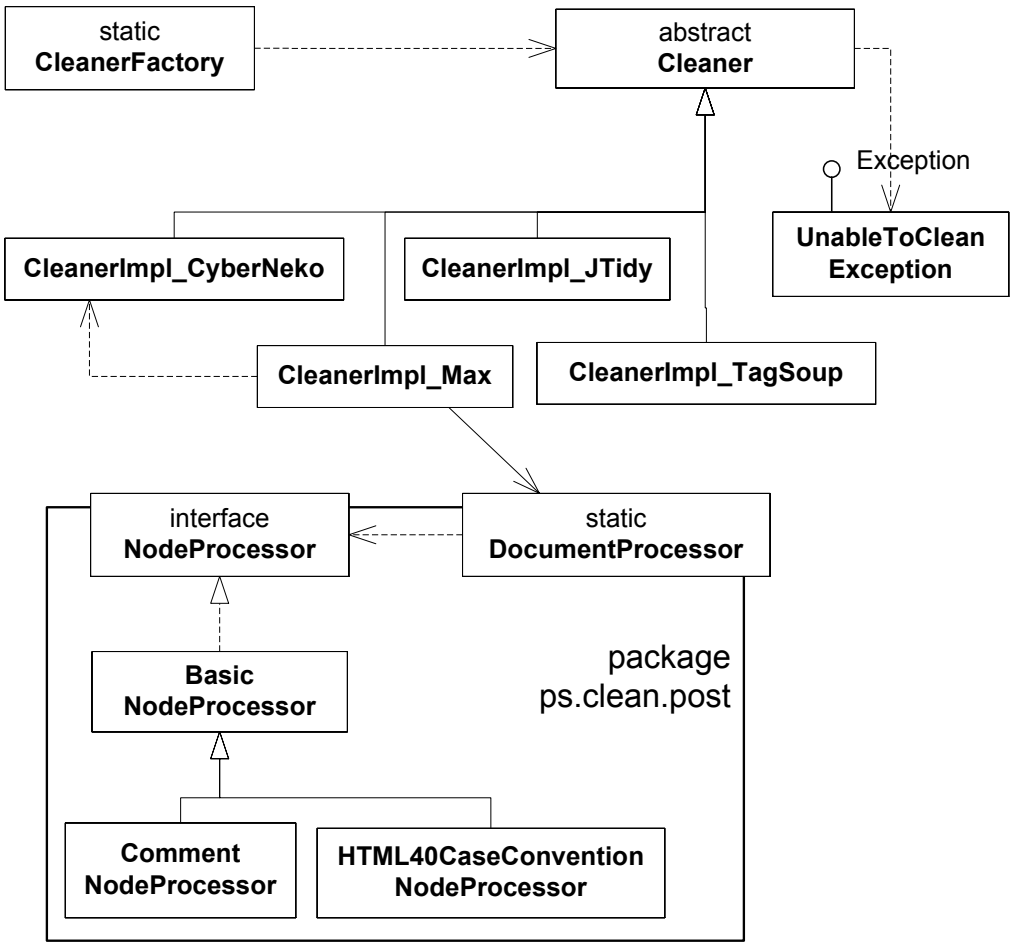


Abbildung 6.4: Klassendiagramm des Moduls *clean*

Nachbearbeitung

Zur Nachbearbeitung der erhaltenen XH-Dokumente wurde ein kleines Rahmenwerk implementiert: Ein statischer `DocumentProcessor` nimmt eine Instanz vom Typ `NodeProcessor` entgegen und kann so Baumelemente beliebig verändern. Die abstrakte Klasse `AbstractNodeProcessor` unterscheidet zwischen den gebräuchlichen Knotentypen „Attribut“, „Kommentar“, „Element“ und „Text“ und delegiert an entsprechende Methoden. Erbende Klassen können so gezielt verschiedene Knotentypen ändern.

Die Klasse `CommentNodeProcessor` repariert z. B. die von *CyberNeko* nicht reparierten Kommentare der Bauart `<!----------------->`. Im HTML 4.0 Standard sind nur Kommentare der Art `<!-- Text -->` zugelassen, also keine weiteren Minuszeichen nach dem Öffnen des Kommentars.

`HTML40CaseConvention` realisiert die standardkonforme Groß- und Kleinschreibung von Element- und Attributnamen.

Anmerkung:

Sprache	Elementnamen und Attributnamen
HTML 4.0	Großbuchstaben
XML	wie in der DTD spezifiziert (Groß- oder Kleinbuchstaben)
XHTML 1.0	Kleinbuchstaben

Tabelle 6.1: Groß- und Kleinschreibung von Element- und Attributnamen nach der DOM-Spezifikation (level 1)

Ein typischer Fehler während der Erstellungsphase war das Zerteilen eines `FORM`-Elementes durch das Modul *clean*: Aus einem `FORM`-Element wurden zwei erzeugt, davon das Zweite allerdings ohne `action`-Parameter. Dadurch kann das HTML-Formular nicht mehr abgesendet werden und hat keine Funktion mehr.

Das Rahmenwerk zur Nachbearbeitung ist eine Erweiterungsmöglichkeit, um bestehende PH-Zerteiler zu verbessern. Diese können unabhängig von der konkret verwendeten Implementierung genutzt werden.

6.1.3 Modul extract

Die häufige Konvertierung von einem Dokumentmodell in ein anderes erwies sich als fehleranfällig, aufwändig zu implementieren und langsam. Da das in d_2c verwendete Dokumentmodell *Dom4j* bereits Methoden zur XSL-Transformation mitbringt, ist es einfachsten und stabilsten diese zu nutzen. Das Extraktionsmodul ist als statische Klasse realisiert (siehe Abb. 6.5) und nutzt die *Dom4j*-Transformationsmethoden. Zu Testzwecken besitzt das Modul *extract* eine `main`-Methode, die es in externen Werkzeugen nutzbar macht.

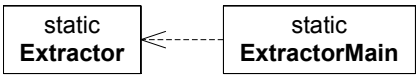


Abbildung 6.5: Klassendiagramm des Moduls *extract*

Probleme der Umsetzung

Die Serialisierung von *Dom4j*-Dokumenten als XML mit der Methode „`.toXML()`“ erwies sich als problematisch, da sie manchmal aufgrund fehlerhaft kodierter Sonderzeichen nicht-wohlgeformte XML-Dateien erzeugt. Eine eigene systemweit verwendete Klasse (`ps.common.DocumentUtils`) sorgt an allen Stellen für eine einheitliche und korrekte Serialisierung von und nach XML-Zeichenketten.

Groß- und Kleinschreibung der Element- und Attributnamen

Da XSLT Groß- und Kleinschreibung beachtet, ist es wichtig, dass die Groß- und Kleinschreibung konsistent und standardkonform ist. Es gibt hier Unterschiede zwischen HTML und XHTML, wie die Tabelle 6.1 zeigt. Eine Nachbearbeitungsklasse im Modul *clean* bringt das XML-Dokument in die gewünschte Schreibweise.

Whitespace

Bei der Extraktion von XML mit XSLT wird *Whitespace* – das sind in XML Leerzeichen, Tabulatoren und Zeilenumbrüche – anders als in PH behandelt. XSLT kümmert sich laut Standard nur um *Whitespace innerhalb* von Elementen. In PH hat aber auch der *Whitespace zwischen* Elementen oft eine Bedeutung. So wird z. B. aus
`<P>Das <B>beste</B> Beispiel</P>` durch XSLT-*Stylesheets*
`<P>Das<B>beste</B>Beispiel</P>` erzeugt. Innerhalb des *Dom4j*-Modells im Speicher sind die Elementknoten noch intakt, so dass durch Nutzung der *Dom4j*-internen XSLT-Transformationsmethoden der *Whitespace* erhalten bleibt.

Namespaces

Die XPath-Ausdrücke in XSLT müssen die *Namespaces* (Namensräume) der XML-Datei beachten. Der in XML definierbare Standard-Namespace gilt *nicht* für die verwendeten XPath-Ausdrücke innerhalb eines XSLT-*Stylesheets*.

Anmerkung:
Das Modul *clean* erzeugt unter Verwendung der *JTidy*-Implementierung mit der XHTML-Option HTML-Elemente mit Namespaces. In der XSLT-Datei muss daher ein Namespace-Präfix, analog zur von *JTidy* erzeugten Definition, deklariert und in den XPath-Ausdrücken genutzt werden.

Innerhalb eines HTML/PH/XH-Dokuments sind *alle* Elemente aus dem gleichen Namespace, d. h. dieser könnte auch ignoriert werden und z. B. durch eine Transformation im Rahmenwerk zur Nachbearbeitung entfernt werden, was aber nicht implementiert ist.

6.2 Entwicklungssystem

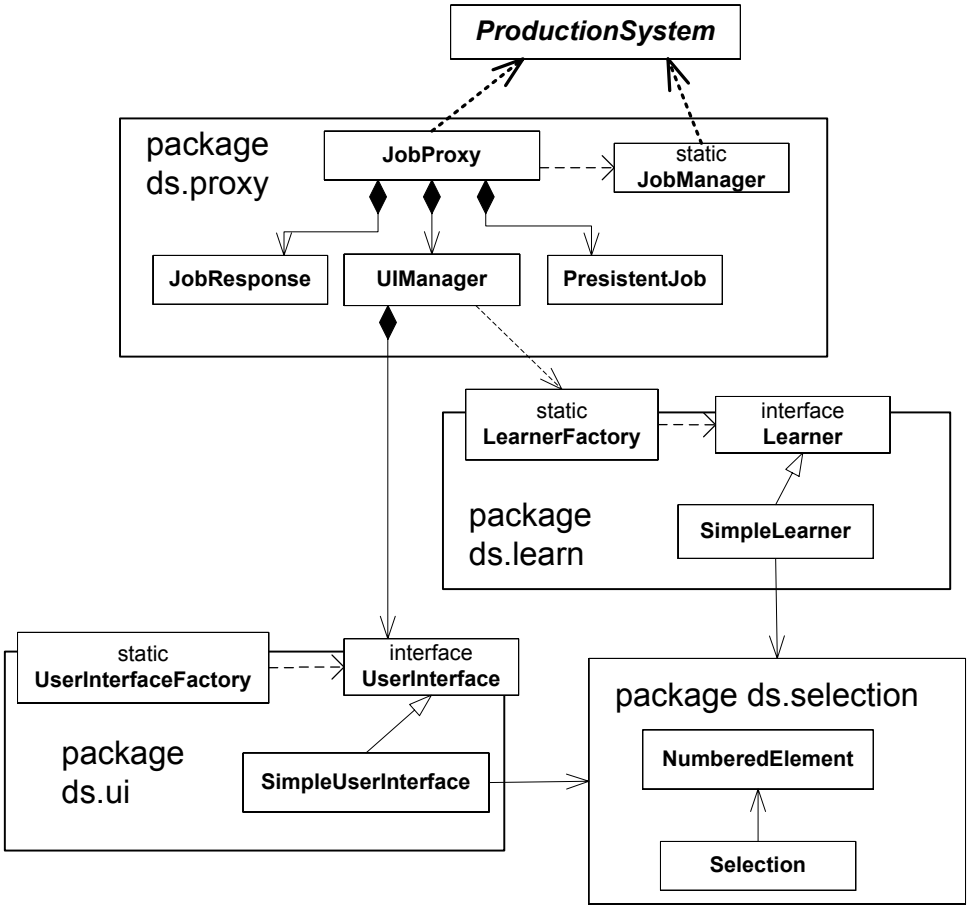


Abbildung 6.6: Abhängigkeiten im Entwicklungssystem

Die Abb. 6.6 gibt einen Überblick über die Paket-Struktur des Entwicklungssystems und seine Kommunikation mit dem Produktionssystem. Die Kopplung zwischen den beiden Ebenen ist nur in den beiden Klassen `JobManager` und `JobProxy` ausprogrammiert. Von den Modulen sind hier nur die wichtigsten Klassen abgebildet, um die Abhängigkeiten zwischen ihnen hervorzuheben. Die drei wichtigen Module (*proxy*, *learn*, *ui*) werden noch detailliert beschrieben.

Das Paket `ds.selection` besteht nur aus datenhaltenden Objekten, die ausschließlich zur Kommunikation zwischen dem Modul *learn* und der Benutzer-Schnittstelle dienen. In den beiden Klassen werden Nummern auf Element-Referenzen abgebildet (`NumberedElement`) sowie Mengen von selektierten Elementen beschrieben (`Selection`).

6.2.1 Modul proxy

Ein Klassendiagramm (6.7) gibt einen Überblick über die Funktionsweise dieses Moduls. Das Modul *proxy* nimmt von außen HTTP-Anfragen über den *Jetty-*

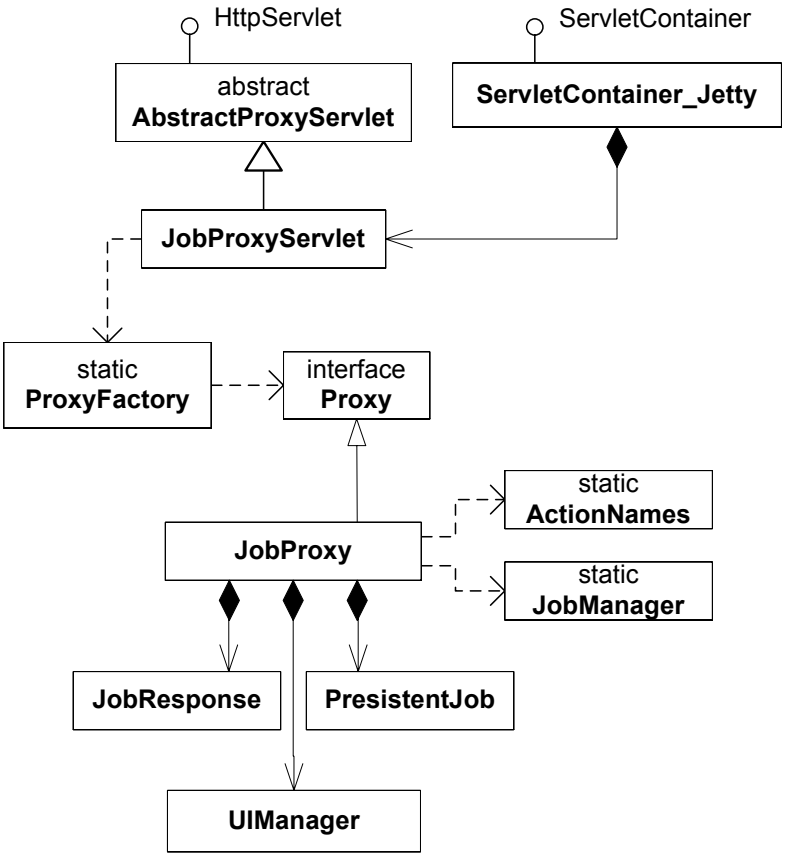


Abbildung 6.7: Klassendiagramm des Moduls *proxy*

Server entgegen. Die Anfragen werden von diesem als `HttpServletRequest` an ein entsprechend konfiguriertes Servlet gesendet. Das `AbstractProxyServlet` konvertiert die Anfrage in ein `JobRequest` und ruft eine Methode der von ihm erbenden Klasse `JobProxy` auf. Diese Klasse muss nur eine einzige Methode implementieren:

```
protected abstract JobResponse handleJobRequest( JobRequest jobRequest );
```

Die Klasse `JobProxy` ist eine zentrale Klasse des Entwicklungssystems, denn sie entscheidet, wie mit der Anfrage des Benutzers umzugehen ist: Soll die Seite nur geladen werden oder ist sie als XH aufzubereiten, um darin Elemente selektieren zu können? Daher ist sie notwendigerweise zustandsbehaftet. Die Klasse kennt drei interne Zustände: *proxy*, *record* und *ui*. Die Abbildung 6.8 zeigt die Zustände und die durch Spezial-URLs und andere HTTP-Anfragen ausgelösten Zustandsübergänge.

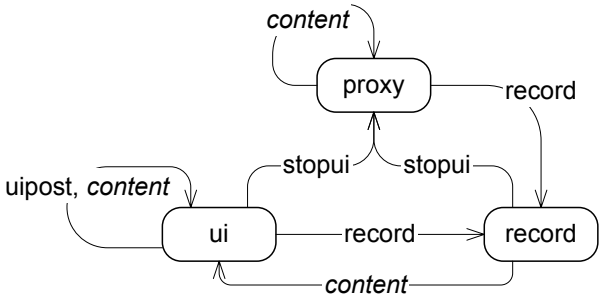


Abbildung 6.8: Zustände des Moduls *proxy*

proxy Grundzustand ist der Zustand *proxy*, in dem die Anfragen mit dem Modul *surf* ausgeführt und beantwortet werden. Eine Aufbereitung nach XH findet also nicht statt.

record Sobald der Proxy in diesen Zustand wechselt ist er „scharf“, d.h. die nächste HTTP-Anfrage nach einer PH-Datei wird als XH aufbereitet und vom Modul *ui* als Selektionswerkzeug umgebaut. Der Proxy wechselt dabei in den Zustand *ui*.

ui Hier kann der Benutzer beliebig lange an der Element-Selektion arbeiten und sie auch zur Bestätigung beliebig oft an den Proxy zurück senden. In diesem Zustand kann die aktuelle Selektion mit einem *job*-Befehl (nicht eingezeichnet) als Wrapper gespeichert werden. Aufrufe der *job*-Befehle ändern nichts am Zustand des Proxys.

content HTTP-Anfrage, die mit dem Inhaltstyp `text/html` beantwortet wird.

Mit der Hilfsklasse `ActionNames` wird die angeforderte URL untersucht. Wenn sie in der virtuellen TLD[2] „d2c“ liegt (z. B. *Name*`.action.d2c`), wird sie als Befehl interpretiert. Alle Befehle zum Speichern, Ausführen oder Testen (Laden der Quellseite) eines *Jobs* (Extraktions-Auftrag) werden an die Klasse `JobManager` delegiert.

Wenn der Benutzer über den Browser den Befehl „Lade die Quellseite des Auftrags“ an den d_2c-Server sendet, muss dieser als Antwort erst eine HTTP-Weiterleitung zur Original-Adresse liefern und anschließend bei der direkt folgenden Anfrage des Browsers aus einem Zwischenspeicher die tatsächliche Quellseite liefern. Nur so kann der Browser die relativen Links auflösen und so z. B. die Bilder einer Seite vom Quell-Webserver laden. Der `JobProxy` hat eine Referenz auf eine `JobResponse`, die er als Zwischenspeicher nutzt.

Während der Benutzer einen Wrapper erstellt, werden die Daten in einem `PersistentJob` gehalten. Dieser kann als ein Verzeichnis serialisiert werden. Mehr über diese Datenstruktur findet sich in Abschnitt 6.3.

[2]Top Level Domain

Probleme der Proxy-Steuerung

Bei der Ansteuerung des Proxys sind einige Probleme zu behandeln, die im folgenden geschildert werden. *Bookmarklets* sind aus Browser-Sicht Verweisziele wie jedes andere auch. Daher speichern Browser (z. B. *Opera*) die HTTP-Antworten auch zwischen. Um dies zu verhindern wird an jeden Befehl zum Proxy eine zufällige Zahl angehängt. Ein derartiges *Bookmarklet* sieht aus wie in Abb. 6.9.

```
javascript:document.location.href=
"http://record.action.d2c/?"+Math.random();
```

Abbildung 6.9: Ein nicht zwischenspeicherbares *Bookmarklet* zur Proxysteuerung

Wenn ein *Bookmarklet* zur Proxy-Steuerung aufgerufen wird, stellt der Browser notwendigerweise eine HTTP-Anfrage. Damit der Browser die aktuell geladene Seite nicht verlässt, wird die HTTP-Antwort des Proxys bei reinen Bestätigungstexten nur in einem kleinen überlagernden Fenster (*Popup-Window*) angezeigt.

Das modifizierte XH-Dokument wird nicht als XML/XHTML, sondern als PH zurück an den Browser gesendet, da manche Browser (z. B. *Internet Explorer*) bei XML/XHTML keine eingebetteten JavaScript-Befehle ausführen. Die Serialisierung als HTML musste angepasst werden, damit Kommentare nicht als `<-...->` ausgegeben werden.

Eine URL der Form

`http://loadname.action.d2c/?kommando=wert`

wird von der *HttpServletRequest*-Klasse falsch interpretiert: Sie findet die Aufrufparameter nicht. Ein Aufrufparameter muss immer an eine Datei adressiert werden und nicht, wie im obigen Fall, an einen Server direkt. Daher müssen die URLs für die Proxy-Steuerung folgenden Aufbau haben:

`http://loadname.action.d2c/etwas?kommando=wert`.

Ebenso müssen Formulare (`<FORM>`) als `action`-Attribut einen Pfad besitzen, der eine Datei referenziert, z. B.

`http://loadname.action.d2c/etwas`.

Zwischenspeicherung (*Caching*)

Um eine BenutzerSchnittstelle im Browser zu realisieren, muss die Zwischenspeicherung des Browsers kontrolliert werden.

Soll eine Seite instrumentalisiert werden, muss der Benutzer zunächst den Proxy in den Aufnahme-Modus versetzen und dann die gewünschte Quellseite anfordern. Normalerweise lädt der Benutzer erst die Seite, für die er einen Wrapper erstellen möchte und aktiviert danach den Aufnahme-Modus. Es ist nun am bequemsten, den „Reload“-Knopf des Browser zu drücken, um die Quellseite anzufordern. Damit der Browser aber wirklich erneut den Proxy anfragt und somit die Selektions-Schnittstelle anzeigt, darf er die Seite nicht aus seinem lokalen Zwischenspeicher laden. Daher müssen vom HTTP-Proxy alle HTTP-Antworten als nicht-zwischenspeicherbar (*uncachable*) umgeschrieben werden. Hierzu sind einige Header zu entfernen und andere zu setzen. Bilder und andere nicht-PH-Dateien dürfen aus Geschwindigkeitsgründen auch zwischenspeicherbar (*cacheable*) bleiben.

Was wird wann gefiltert?

Filtern bezeichnet im folgenden Text die Manipulation von HTTP-Headern. HTTP-Nachrichten, die gefiltert werden, sollen auch stets protokolliert werden.

- Binäre Daten sollen weder gefiltert noch protokolliert werden, dürfen aber zwischengespeichert werden. Die HTTP-Header dürfen also nicht verändert werden.
- HTTP-Antworten mit HTML-Inhalt sind zu filtern und zu protokollieren. Damit kann unter anderem das *Session Management* analysiert werden. Eine HTTP-Antwort, welche die Startseite eines Fernsteuerungs-Dienstes enthält (siehe 2.4.2), muss nicht gefiltert werden. Erst die nun folgenden Anfragen und Antworten sind interessant.
- HTTP-Weiterleitungen (Code 302) enthalten oft keine Angabe über den Inhaltstyp (`Content-Type`). Sie werden nicht gefiltert, aber protokolliert.
- Auch HTTP-Antworten, die nur JavaScript-Anweisungen enthalten (z. B. *Dateiname*`.js`), dürfen nicht aus Versehen instrumentalisiert werden.

Zum Erstellen eines Wrappers sendet der Benutzer zunächst über ein *Bookmarklet* eine Anfrage für die URL `record.action.d2c` an den Proxy. Dieser soll daraufhin die nächste HTML-Seite instrumentalisieren.

Den Inhalt der nächsten HTTP-Anfrage zu instrumentalisieren genügt nicht: Wenn der Benutzer, während die Seite lädt (z. B. eingebettete Bilder), den Modus `record` aktiviert, so würde der Proxy versuchen, eine HTTP-Anfrage für eine Grafik mit einer instrumentalisierten Version derselben zu beantworten!

Daher wird zum bestimmen des Inhaltstyps der in 6.10 angegebene Algorithmus verwendet.

```
IF Header Content-Type vorhanden
  IF startsWith text/html
    THEN return HTML
    ELSE return OTHER // anderer Content-Type
  ELSE // Header "Content-Type"nicht vorhanden
    return HTML // Heuristik
```

Abbildung 6.10: Pseudocode für die Bestimmung des Inhaltstyps einer HTTP-Antwort

6.2.2 Modul learn

Dieses Modul hat die Aufgabe, aus der mit dem Modul *ui* erstellten Selektion auf einem XML-DOM das extrahierende XSLT-*Stylesheet* zu erzeugen. Dazu werden die selektierten Elemente jeweils mit einer `<xslt:copy>`-Funktion in neu angelegte `<item>`-Elemente verpackt. Ein solches XSLT-*Stylesheet* ist in Abb. 6.11 dargestellt, ein Ergebnis ihrer Anwendung ist auf Seite 75 zu sehen.

```
<xsl:template match="\">
  <d2cExtractionResult>

Für jedes zu extrahierende Element:
    <item>
      <xsl:copy-of select="XPath-Ausdruck"/>
    <item>

  </d2cExtractionResult>
</xsl:template>
```

Abbildung 6.11: Aufbau eines extrahierenden XSLT-*Stylesheets*

Um eine leichte Erweiterbarkeit des Entwicklungssystems sicherzustellen, wurde auch in diesem Modul der Ansatz (siehe Abb. 6.12) statische Fabrik (`LearnerFactory`), öffentliche Schnittstelle (`interface Learner`) und konkrete Implementierung (`SimpleLearner`) gewählt.

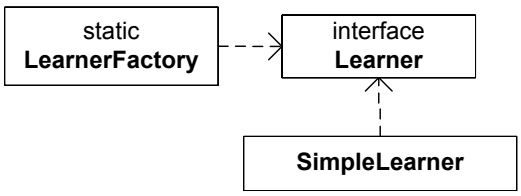


Abbildung 6.12: Klassendiagramm des Moduls *learn*

Anmerkung:
Bei der Implementierung ist darauf zu achten, dass die Element- und Attributnamen konsistent in Groß- oder Kleinschreibung vorliegen, damit die XPath-Ausdrücke im XSLT-*Stylesheet* ebenfalls konsistent funktionieren. Die korrekte Schreibweise wurde durch eine Vorverarbeitung des DOMs erreicht.

6.2.3 Modul ui

Dieses Modul kapselt Funktionalität zur Interaktion mit dem Browser. Dazu wird die vom Proxy gelieferte XH-Seite auf zwei Weisen instrumentalisiert: Im Header wird ein JavaScript-Block eingebaut, der von *Bookmarklets* und `onClick-Handlern` aktiviert werden kann. Jedes selektierbare Element erhält eine eindeutige Nummer als Attribut und einen `onClick-Handler`, der eine JavaScript-Funktion im Header aufruft. Der JavaScript-Block im Header hebt selektierte Elemente über dynamische CSS-Anweisungen hervor. Die Integration von JavaScript-Anweisungen in Webseiten ist aus Sicht der Benutzer beliebter als das Einfügen von Java-Applets oder HTML-Elementen [FGC+97].

Instrumentalisieren bestehender PH-Seiten

Da andere Seiten bereits JavaScript-Programme enthalten, die nicht angetastet werden sollen, darf der eingefügte Programmtext nicht den bestehenden überschreiben oder versehentlich aufrufen. Daher beginnen alle globalen Variablen und Funktionsnamen mit einem frei wählbaren Präfix. Im Folgenden wird der fixe Präfix „`d2c_`“ verwendet.

Klassenstruktur

Die Klasse `SimpleUserInterface` ist die zentrale Klasse dieses Moduls (siehe Abb. 6.13). Sie verwaltet das Original-Dokument und dazu als „Zwillings-

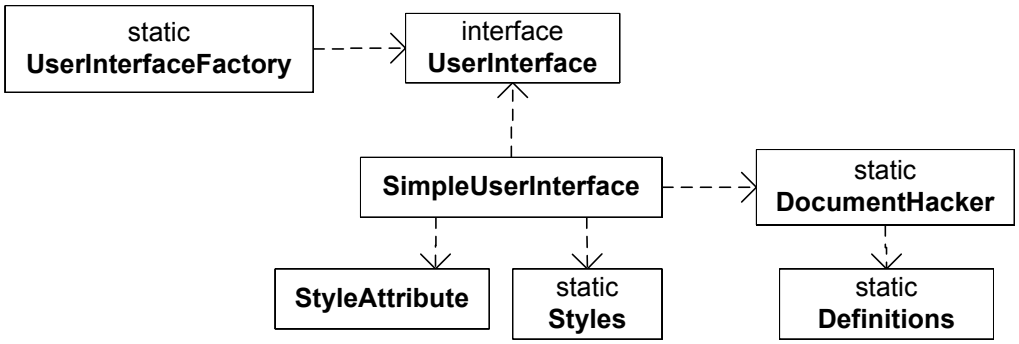


Abbildung 6.13: Klassendiagramm des Moduls *ui*

dokument“ eine durch die Hilfsklasse `DocumentHacker` modifizierte Version. In `Definitions` steht lediglich, welche HTML-Elemente einen `onClick`-Handler haben dürfen und welche davon (Standard: alle) anklickbar sein sollen. Die Klasse `StyleAttribute` abstrahiert von den CSS-Angaben. Sie kann diese zerteilen, manipulieren und wieder als Zeichenkette ausgeben. Die Definition von selektierten und unselektierten Elementen befindet sich ausgelagert in `Styles`.

Schritte der Instrumentalisierung

Das Instrumentalisieren einer PH-Seite läuft in folgenden Schritten ab (siehe dazu Abb. 6.14: Im `HEAD` des HTML-Baumes wird ein JavaScript-Block eingebaut (1). Dort wird eine Funktion eingebaut, die für selektierte Elemente jeweils alle Eltern bis zur Wurzel und alle Kinder bis zum Blatt deselektiert. Der Selektions-Status eines Elements wird mit DOM-Funktionen in einem Attribut des Elements gespeichert. Jedes Element erhält eine eindeutige Nummer im Attribut `d2c_uid` (2). Des weiteren erhält jedes Element, für welches dies laut HTML 4 erlaubt ist,

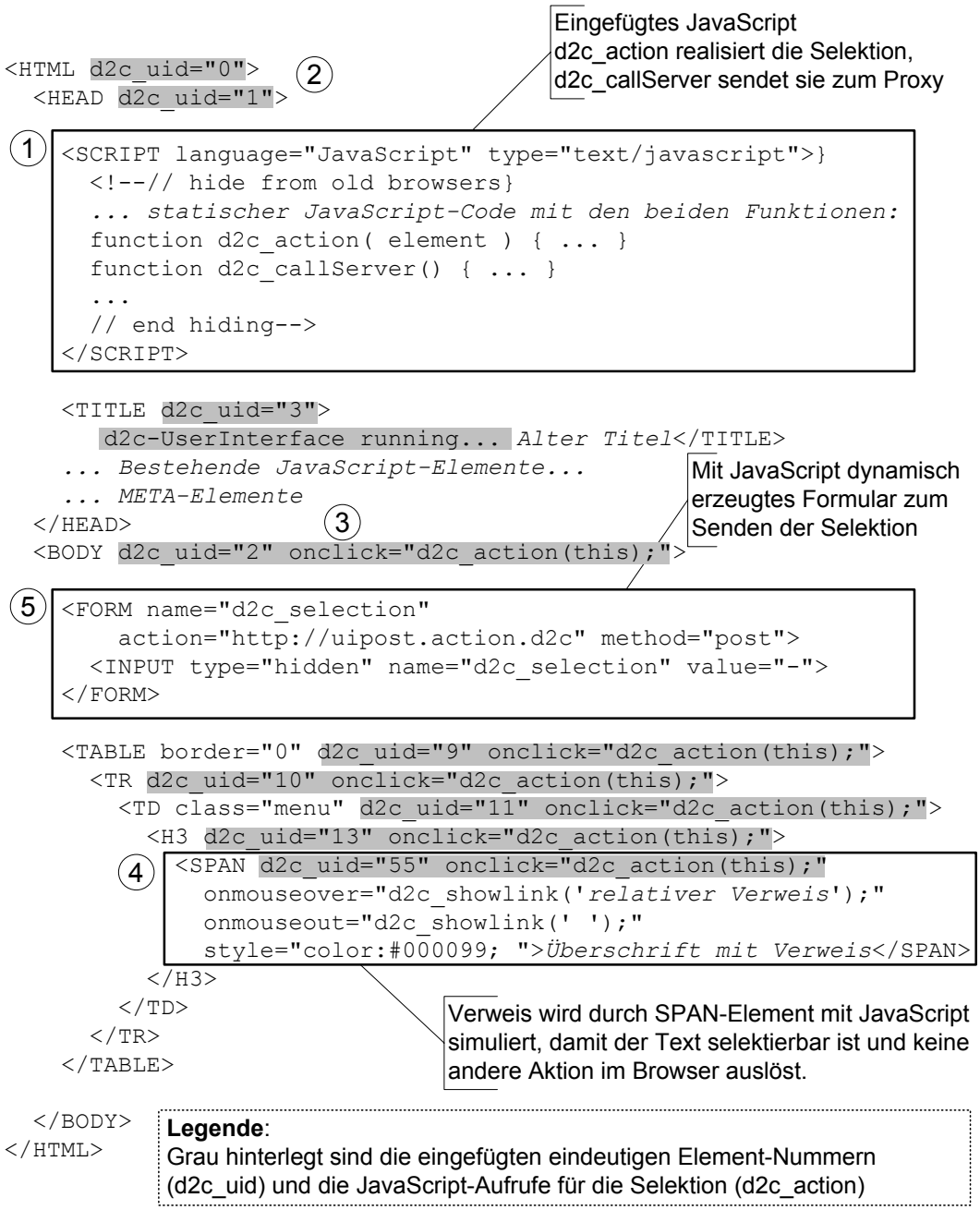


Abbildung 6.14: Eine instrumentalisierte XH-Seite

einen onClick-Handler mit `d2c_action(this)` (3). Dadurch wird im statischen JavaScript-Bereich eine Referenz auf das angeklickte Element übergeben. So kann dann die eindeutige Nummer ausgelesen werden. Verweise (`<A HREF="...">`) werden durch `<SPAN>`-Elemente ersetzt und wie Verweiselemente dargestellt (4). Damit der Browser zudem beim Überfahren mit der Maus das Verweisziel in der Statuszeile anzeigt, wird mit einem `onMouseOver`-JavaScript dieses Verhalten nachgebildet. Zum Absenden der Selektion wird mit JavaScript dynamisch ein `FORM`-Element mit den notwendigen Attributen in die Seite eingebaut (5). Das dynamische Einbauen in das Browser-DOM ist robuster als eine serverseitige Manipulation. Anschließend wird mit Tiefensuche der gesamte HTML-Baum abgesucht und jedes selektierte Element zu einer kommagetrennten Zeichenkette hinzugefügt. Diese wird mit einem HTTP-POST an den Proxy übertragen.

Nach der Manipulation im Nutzdatenteil der HTTP-Antwort muss der „`Content-Length`“-Header an die veränderte Dokumentlänge angepasst werden.

Färben der instrumentalisierten Seite

Der Versuch, jedem Element ein 10 %-transparentes PNG[3] zu geben und dem Benutzer über die verschiedenen Färbungen optisch intuitiv zu zeigen, wie die Seite aus Elementen aufgebaut ist, scheitert an mangelhaften PNG- und CSS-Implementierungen der Browser: Der *Internet Explorer* kann noch keinen PNG-Alpha-Kanal und *Opera* stellt die mit CSS gesetzten Hintergründe von Elementen nicht richtig dar.

6.3 Gemeinsame Module

Die Abbildung 6.15 zeigt die gemeinsam genutzten Klassen, deren Aufgabe kurz beschrieben wird:

Die Klassen `JobRequest` und `-Response` sind die systemweit verwendeten Abstraktionen der HTTP-Anfragen und -Antworten. Sie sind als Unterklassen der abstrakten `HttpCommonMessageFields` implementiert, welche die gemeinsamen Anteile kapselt. Die beiden Job-Klassen sind notwendig, da die naheliegenderweise zu verwendenden Klassen `HttpServletRequest` und `HttpServletReponse` einige Probleme aufweisen:

- Nach dem Lesen von HTTP-Parametern kann der Nutzdatenteil (Body) der HTTP-Antwort nicht mehr gelesen werden.
- Der Nutzdatenteil kann aufgrund der Datenstrom-orientierten Verarbeitung nur einmal gelesen werden.

Daher erben die Job-Klassen nicht von den entsprechenden *Servlet*-Klassen sondern sind als eigene datenhaltende Klassen mit zahlreichen problemspezifischen Zusatzmethoden implementiert. Die nötige Umwandlung leistet die Klasse `ds.proxy.JobServletConversion` aus dem Modul *proxy*.

[3] Portable Network Graphics, der Nachfolger von GIF

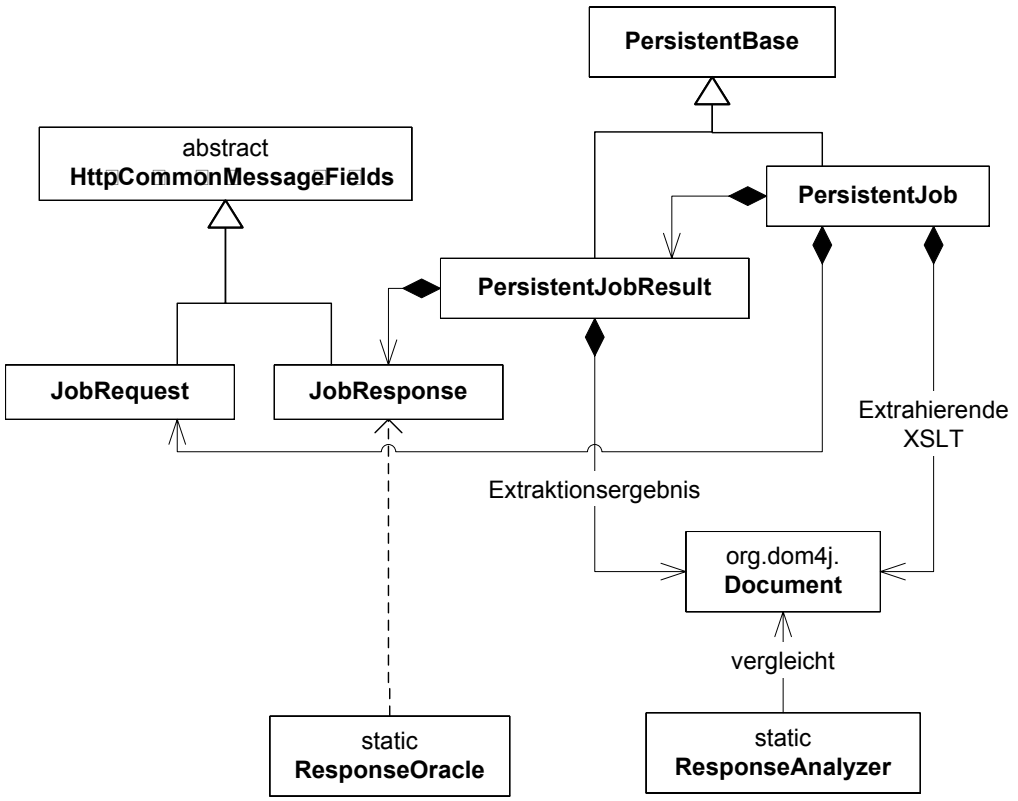


Abbildung 6.15: Klassendiagramm der gemeinsamen Module

Job (Extraktions-Auftrag)

Mit Job wird die Datenstruktur zur Kommunikation zwischen dem Entwicklungssystem, das Jobs erstellt und dem Produktionssystem, das sie ausführt, bezeichnet. Ein Job enthält eine HTTP-Anfrage und eine (von d_2c erstellte) XSLT-Transformation. Nach Ausführen eines Jobs enthält dieser zudem das Extraktionsergebnis. Ein Job wird jeweils in einem Verzeichnis in mehrere XML-Dateien serialisiert. Die innerhalb von JavaScript vorkommenden '-Zeichen werden als `&apos;` kodiert, wodurch die serialisierten XH-Seiten mit JavaScript nicht in einem Browser ausführbar sind.

Zur Implementierung nutzt die Klasse `PersistentJob` die `PersistentJobBase`, welche von der Verzeichnisstruktur im Dateisystem abstrahiert. Die Klasse `PersistentJob` selbst kapselt die Serialisierung von und nach XML-Dateien. Dazu nutzt sie die Klasse `PersistentJobResult`, die wiederum ein Extraktionsergebnis als Dokument speichert. Zusätzlich hält sie auch eine Referenz auf eine `JobResponse`, und zwar auf die Original-Seite, aus der die Daten extrahiert wurden, wodurch potentiell eine automatische Reparatur des Wrappers möglich wird (siehe 8.2.5). Die Klasse `ResponseOracle` implementiert die in 6.10 beschriebenen Bestimmung des Inhaltstyps einer HTTP-Antwort. In `ResponseAnalyzer` ist eine rudimentäre Vergleichsmöglichkeit für das vorangehende und aktuelle

Extraktionsergebnis vorhanden. Die in 8.2.4 geplante Erweiterung wird hier implementiert.

Fehlerbehandlung

Es wurde in den Modulen des Produktionssystems jeweils eine eigene Fehlermeldung spezifiziert. So müssen andere Module nur einen Fehler abfangen und behandeln, wodurch sie austauschbarer werden und das System damit flexibler bleibt. Eine Behandlung von Fehlern eines Moduls in einem anderen würde das Modulgeheimnis verletzen und eine Austauschbarkeit von Modul-Implementierungen unmöglich machen. Der Text der Fehlermeldung gibt dem Entwickler weiterhin Informationen über den eigentlichen Fehler.

Logging

Zur Programmablauf-Protokollierung (Logging) wird das *open source*-Paket *Log4j* verwendet. Es lässt sich komfortabel über eine Konfigurationsdatei einstellen, unterstützt verschiedene Wichtigkeitsstufen von Meldungen und ist zudem recht schnell.

Allgemeine Probleme mit der Kodierung

Die Kodierungsangaben von HTTP und HTML sind kompliziert und teilweise widersprüchlich:

- HTTP-Header benutzen fix die Kodierung *US-ASCII*.
- Einer der Header (`Content-Type`) kann die Kodierung der Nutzdaten in der HTTP-Antwort angeben.
- Ebenso kann *innerhalb* einer HTML-Datei in einem META-Tag (`HTTP-equiv`) die Kodierung angegeben werden. Diese Information kann allerdings nur gelesen werden, *nachdem* die Kodierung bereits bekannt ist. HTTP-Clients können hier nur Heuristiken verwenden, um die Kodierung zu erraten, die am Besten zu passen scheint.
- Es kommen in der Praxis auch gänzlich fehlende Informationen vor, z. B. bei HTTP-Weiterleitungen (Code 302), die trotzdem einen HTML-Datenkörper besitzen.
- Ebenso müssen bei einer HTTP-Anfrage die HTTP-GET/POST-Parameter korrekt kodiert sein, allerdings besteht z. B. für GET-Parameter kein Standard.

Kodierung in d_2c

d_2c nutzt das Java-interne, sehr mächtige Unicode als Standard-Codierung. Unicode ist die umfangreichste Kodierung, die in Java zur Verfügung steht. Für die Ausgabe zum Browser oder als serialisiertes XML wird auch konsequent *eine* Kodierung benutzt: `ISO-8859-1`, auch bekannt als `latin-1`. Es ist auf westlichen Computersystemen mit fast allen Texteditoren bequem zu bearbeiten – was für Unicode leider nicht gilt.

6.4 Statistik

Eine kleine Übersicht (Tabelle 6.2) über die verschiedenen Module zeigt, dass z. B. die Implementierung des Benchmarks (Paket `app`) aufwändiger ist als die Implementierung des Entwicklungssystem. Das Produktionssystem ist vor allem durch die systemweit genutzten datenhaltenden Klassen (`ps.commons`) so groß.

Modul	Pakete	Klassen	Quelltextzeilen	echte Quelltext-zeilen (rlosc)
ps.commons	3	18	3315	1312
ps	1	2	417	200
ps.surf	1	7	583	241
ps.clean	2	14	698	310
ps.extract	1	3	191	104
ps.* (ohne ps.commons)	5	26	1889	855
ps	9	44	5204	2167
ds	11	28	2835	1216
app	5	29	3019	1396
ws	3	3	224	92
d_2c-Gesamt	28	104	11282	4871

Tabelle 6.2: Quelltext-Statistik

7 Evaluation

In diesem Kapitel wird das Gesamtsystem aus Sicht des Benutzers evaluiert. Zunächst wird die Erstellung eines Wrappers im Detail beschrieben, anschließend seine Nutzung zu Test- und Einsatzzwecken.

7.1 Benutzer-Schnittstelle

7.1.1 Erstellen eines Wrappers

Anhand einer bekannten und komplexen Webseite (hier: *CNN*-Homepage) wird in allen Schritten demonstriert, wie ein Wrapper mit wenigen Mausklicks erstellt wird. Dabei wird ein ausführlicher Weg begangen, der *jede* Aktion des Nutzers beschreibt.

1. Schritt: Surfen

Der Benutzer trägt d_2c als Proxy ein und bewegt sich dann wie gewohnt im Internet. Dabei findet er eine Seite, aus der er Daten extrahieren möchte: Die *CNN*-Homepage (Abb. 7.3 A, auf der folgenden Seite).

2. Schritt: Aufnahme-Modus aktivieren

Der Benutzer ruft das *Bookmarklet* `record` über seine Lesezeichen auf. In einem neuen (kleinen) Fenster zeigt d_2c an, dass es nun die nächste Anfrage als Quellseite betrachten wird (siehe Abb. 7.1).

Abbildung 7.1: Proxy ist bereit zur Extraktion

3. Schritt: Quellseite laden

Der Benutzer drückt den *Reload*-Knopf seines Browsers, und lädt damit die *CNN*-Seite erneut. Der Browser stellt eine modifizierte Version der PH-Seite dar. Der Benutzer kann nur am Titel erkennen, dass die Seite ein Selektionswerkzeug ist. Der veränderte Titel lautet „`d2c user interface running...` *alter Titel*“ (siehe Abb. 7.2). Alternativ hätte der Benutzers auch eine beliebige URL in die Adresszeile des Browser eingeben und laden können. Um die Ergebnisse von Suchanfragen als Quellseite zu verwenden, sendet der Benutzer an dieser Stelle das ausgefüllte Formular für die Suchanfrage ab.

Abbildung 7.2: Anzeige der Modifikation nur im Titel

4. Schritt: Elemente im Browser auswählen

Der Benutzer wählt alle für ihn interessanten Elemente einfach durch Anklicken aus, die daraufhin einen kräftigen roten Rand erhalten. Erst wird das kleinstmögliche Element selektiert, weiteres Klicken auf die gleiche Stelle vergrößert die Selektion (siehe Abb 7.3 B,C,D). Klicken auf andere Elemente fügt diese zur Selektion hinzu (siehe Abb 7.3 E). Während der Selektion findet keine Kommunikation mit dem Server statt.

5. Schritt: Selektion zum Server senden

Über das *Bookmarklet* `select` wird die Selektion an den Server gesendet. Zur Bestätigung antwortet dieser mit einer Version der Seite, auf der die selektieren Elemente grün eingefärbt sind (siehe Abb. 7.3 F).

Anmerkung:
Die Schritte (B) bis (F) können solange wiederholt werden, bis die grün gefärbte Selektion die vom Benutzer gewünschten Elemente enthält. Üblicherweise ist nur ein einziger Durchgang notwendig.

6. Schritt: Speichern des Wrappers

Der Benutzer wählt das *Bookmarklet* `save`, um den Wrapper zu speichern. Der Proxy zeigt daraufhin eine Liste mit bereits gespeicherten Wrappern und ein Textfeld für die Eingabe eines neuen Namens (siehe Abb 7.4 A).

Der Benutzer klickt auf einen der alten Namen oder gibt einen neuen ein und bestätigt mit `OK`. Der Server bestätigt den erfolgreich erzeugten Wrapper mit einer Meldung (siehe Abb. 7.4 B).

7. Schritt: Ende der Erstellung

Zuletzt versetzt der Benutzer durch Aufruf des *Bookmarklets* `stopui` wieder in den Normalzustand. Dieser Befehl ist auch während der anderen Schritte anwendbar, um die Wrapper-Erstellung vorzeitig abzubrechen.

7.1.2 Testmöglichkeiten

Drei Möglichkeiten stehen zur Verfügung, um die erstellten Wrapper zu testen. Sie werden über gleichnamige *Bookmarklets* genutzt.

„Ansurfen“ (`surf`) führt nur die gespeicherte HTTP-Anfrage aus und zeigt das Ergebnis im Browser an. So kann manuell verifiziert werden, dass die Webseite noch wie erwartet aussieht.

„Extrahieren“ (`extract`) führt den Wrapper aus und stellt das Extraktionsergebnis im Browser auf einer Seite dar; sowohl als XML-Quelltext und zusätzlich in gewohnter Browser-Darstellung, wodurch komplexe Extraktionsergebnisse be-

(A) unverändert

(B) erster Klick selektiert Element

(C,D) mehrmaliges Klicken vergrößert Selektion

(E) fertige Auswahl

(F) Server bestätigt Selektion

Abbildung 7.3: Selektionsschritte am Beispiel der *CNN*-Homepage

(A) Dialog für einen Dateinamen (B) Ein Wrapper wurde erzeugt

Abbildung 7.4: Speichern des Wrappers

quem im Browser inspiziert werden können.

„Ausführen“ (`run`) führt auf Wunsch *alle* bisher erstellen Wrapper aus und schreibt die Ergebnisse auf die Festplatte. Wenn einer der Wrapper fehlerhaft ist, erscheint eine entsprechende Meldung.

7.1.3 Integration der Wrapper

Unter der URL
`http://`*d_2c-Server*`/d2c/foo?d2c_jobname=`*Jobname*
Optional: `&param1=`*Wert*`&param2=`*Wert*
liefert das System das Extraktionsergebnis als XML zurück. Werden URL-Parameter angegeben, so nutzt das System sie zur Laufzeit, um die Anfrage des Wrappers entsprechend abzuändern.

Als Beispiel wird das Ergebnis des oben erstellten *CNN*-Wrappers gezeigt (Abb. 7.5).

Anmerkung:
Das Abarbeiten von Job ist schnell genug für den Einsatz in darauf aufbauenden Systemen. So dauert z. B. das Abarbeiten von 30 Jobs bei einem Rechner mit 1,13 GHz und einer DSL-Internetanbindung (768 KB/s) knapp 2 Minuten, also nur 4 Sec. pro Job. Da dieser Wert akzeptabel ist, wurden keine weiteren Messungen und keinerlei Optimierungen durchgeführt.

7.2 Durchgeführte Extraktionen

Für einige der zu Anfang der Arbeit genannten Beispiele wurden testweise Wrapper erzeugt. An dieser Stelle soll kurz über Erfahrungen mit ihnen berichtet werden.

Folgende Wrapper wurden vor Abschluß der Entwicklungsphase im Zeitraum vom 7.-10. Februar 2003 erzeugt und lieferten bis zum Ende des Tests (18. April 2003) korrekte Ergebnisse: „Nachrichtenticker“, „Schneehöhen-Bericht“, „Vorlesungs-Homepage“, „MP3-Datenbank“ und „Wettervorhersage“. Drei weitere erstellte Wrapper mussten zwischenzeitlich repariert werden: Bei „Telefonbuch.de“ und „Ebay.de“ aufgrund von internen Änderungen und bei „Amazon.de“ wegen einer veränderten Quellseite. Später wurden noch Wrapper für die Cnn-Homepage, den „Google-Zähler“ und die „Zufällige Url“ erstellt. Alle Wrapper konnten innerhalb weniger Minuten über die Wysiwyg-Oberfläche im Browser erstellt werden, lediglich bei der Wettervorhersage musste manuell der richtige *Frame* ermittelt und im Browser eingeben werden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PersistentJobResult>
  <d2cExtractionResult>
    <item>
      <DIV class="cnnMainT1">
         ... gekürzt ...
      </DIV>
    </item>
    <item>
      <DIV class="cnnFinePrint" style="color: #333; padding:6px;
      padding-left:0px;">Updated: 05:10 p.m. EDT (2110 GMT) April
      10, 2003</DIV>
    </item>
    <item>
      <DIV class="cnnIndexCaption">Kurds celebrate Thursday as
      they drive through the northern Iraqi town of Khanaqin.
      </DIV>
    </item>
  </d2cExtractionResult>
  <Status>
    <Text>STATUS_PAGE_STRUCTURE_CHANGED_EXTRACTION_BROKEN
      </Text>
    <Numeric>11</Numeric>
  </Status>
  <RequestUrl>http://www.cnn.com/</RequestUrl>
</PersistentJobResult>
```

Abbildung 7.5: Extrahiertes Xml aus der *CNN*-Seite

7.3 Beurteilung anhand der Kriterien

Anhand der auf Seite 14 dargestellten Kriterien wird das erzeugte System hier gemessen.

Eigenschaften erzeugter Wrapper

K 1a Durch sorgfältige Auswahl und Verwendung von zuverlässigen Komponenten für die Kommunikation mit Webservern und das Aufbereiten ihrer Daten konnten für alle Aufgaben des letzten Abschnittes Wrapper erzeugt werden. Dabei konnten die gewünschten Inhalte erfolgreich extrahiert werden. Bei Seiten, die *Frames* enthalten, muss der Benutzer manuell die Adresse des gewünschten *Frames* in den Browser eingeben. Das Kriterium K 1a ist somit erfüllt.

K 1b Es wurde ein Benchmark entwickelt, um den PH-Zerteiler zu finden, der die Aufbereitung von PH am ähnlichsten zu einem marktbeherrschenden Browser (hier: *Opera*) durchführt. Dadurch konnte Kriterium K 1b erfüllt werden.

Erstellung von Wrappern

K 2 d_2c ermöglicht als erstes System seiner Art die vollständige Erstellung eines Wrappers in WYSIWYG-Art im Browser. Wrapper können schnell und ohne Programmier- oder HTML-Kenntnisse auch von Laien erzeugt werden. Die gesamte Benutzer-Schnittstelle läuft ausschließlich im Browser ab, um für den Benutzer Kontextwechsel zu vermeiden. Dadurch ist K 2 erfüllt.

Kompatibilität und Wiederverwendung

K 3a Zur einfachen Integration in bestehende Systeme wird das Extraktionsergebnis als XML über eine Webdienst-Schnittstelle ausgegeben. Damit ist K 3a erfüllt.

K 3b Durch die modulare Architektur lassen sich wichtige Komponenten leicht austauschen, dadurch profitiert das System zeitnah von verbesserten Implementierungen. Auch die Weiterentwicklung des Gesamtsystems ist durch Schichten- und Modulkonzepte leichter möglich. Es werden bestehende Komponenten und Standards verwendet, wodurch der Benutzer auch bestehende Werkzeuge benutzen kann. Insbesondere werde´n die wichtigen Standards XML und XSLT zur Baumbeschreibung und -transformation verwendet. Das Kriterium K 3b ist erfüllt.

7.4 Anwendungsmöglichkeiten

Das entwickelte System kann primär dazu dienen, viele Wrapper zu erstellen und aktuell zu halten. Da die komplette Schnittstelle im Browser abläuft, kann d_2c auch leicht als Service eines Internet-Portals (z. B *Yahoo*) angeboten werden. Einige nahe liegende Anwendungsmöglichkeiten des Systems sind:

- Meta-Suchmaschinen aller Art sind ein ideales Anwendungsfeld, da hier viele Wrapper gepflegt werden müssen. Da die Wrapper-Erstellung selbst von Laien durchgeführt werden kann, sind personalisierte Meta-Suchmaschinen denkbar, bei der jeder Benutzer sich die Quellen seiner Meta-Suchmaschine individuell mit *d2c* zusammenstellt.
- Ein WAP[1]-Telefonbuch. *d2c* extrahiert aus einer Telefonauskunfts-Seite dynamisch die Telefonnummer für einen zu suchenden Nachnahmen. Per XSLT kann das Extraktionsergebnis dann in WML[2] transformiert und auf das Mobiltelefon des Benutzers gesendet werden.
- Auch die in der Einleitung angesprochenen Preisvergleichs-Dienste lassen sich mit *d2c* leicht realisieren, da hier viele Wrapper aktuell gehalten werden müssen, ähnlich wie bei einer Meta-Suchmaschine.
- Interessant könnte auch ein Dienst sein, der auf einem Handelsplatz wie *Ebay* den Marktwert eines Artikels über einen längeren Zeitraum ermittelt, indem die Ergebnisse aller Auktionen, in denen ein solcher versteigert wird, extrahiert und gemittelt werden.

Eine zentrale Eigenschaft des Systems ist seine Fähigkeit, als Proxy die meisten HTML-Seiten zu modifizieren, ohne sie in ihrer Funktionsweise zu beeinträchtigen. Dadurch lassen sich Webseiten zusätzlich mit Informationen anreichern und folgende Anwendungen des filternden Proxys sind denkbar:

- Abkürzungen werden beim Überfahren mit der Maus durch einen kurzen Text erklärt. Die Erklärungstexte können mit Hilfe von *d2c* aus der Webseite eines Abkürzungswörterbuches extrahiert werden.
- Werbebanner könnten über eine Heuristik entfernt werden.

Die in dieser Arbeit implementierte Möglichkeit, HTML-Elemente im Browser zu selektieren, kann z. B. in *Usability*-Studien eingesetzt werden, indem bestehende Webseiten so instrumentalisiert werden, das jeder Klick des Benutzer registriert wird. So kann das Navigationsverhalten *auf beliebigen, bestehenden Webseiten* analysiert werden.

Auch ein WYSIWYG-HTML-Editor zur Farbgestaltung ist auf Grundlage der Selektionsoberfläche denkbar. Der Benutzer kann auf der HTML-Seite im Browser die Farben verändern, bis sie ihm gefallen und die gewählte Farbkombination z. B. als Email erhalten.

[1] Wireless Application Protocol
[2] Wireless Markup Language, eine Art minimal-HTML für Mobiltelefone

8 Vorgesehene Erweiterungen

In diesem Kapitel werden Erweiterungen beschrieben, welche mit geringem weiteren Aufwand den Nutzen deutlich steigern können. Es ist geplant *Frames* und *SSL* transparent zu behandeln. Eine automatische Erkennung der gemeinsamen Struktur einer Menge von Webseiten ist in zweierlei Weise nützlich. Zum einen können die statischen Elemente von der Selektion ausgeschlossen werden und diese so vereinfachen. Auch eine automatische Extraktion *aller* dynamischen Inhalte ist denkbar. Zum anderen können alle gemeinsamen Elemente als Rückversicherung benutzt werden, wodurch eine Wrapper-Validierung ermöglicht wird. Eine Ergebnis-Analyse soll zusätzlich zu Wrapper-Validierung die Unterschiede vom aktuellen zum letzten Wrapper-Ergebnis ermitteln. Alle genannten Erweiterungen verwenden den bereits im Abschnitt 5.3 definierten *größten gemeinsamen Schnittbaum* von zwei XML-Dokumenten.

8.1 Abdeckung der Anwendungsdomäne

Obwohl d_2c bereits für einen sehr großen Teil von Webseiten und Webservern Wrapper erzeugen kann, könnte durch einige Erweiterungen der Abdeckungsgrad weiter erhöht werden.

SSL

Sowohl *HttpClient* als auch *Jetty* (siehe Kapitel Wiederverwendung) unterstützen prinzipiell auch SSL[1]. Es ist allerdings aufwändig zu konfigurieren (Schlüsselmanagement) und wird daher in der aktuellen Version von d_2c nicht genutzt, wodurch d_2c z. B. beim Anmelden auf `web.de` scheitert.

Frames

Zur Zeit ist lediglich eine Warnmeldung implementiert, wenn der Benutzer versucht aus einem *Frameset* Daten zu extrahieren. Zur Behandlung von *Frames* ist vor allem ein guter fehlertoleranter PH-Zerteiler notwendig, um die *Frameset*-Definition in jedem Fall korrekt zu interpretieren und jeden Unter-*Frame* zu instrumentalisieren. Da meistens auch JavaScript zur Navigation zwischen *Frames* benutzt wird, ist die Ausführung von JavaScript eine notwendige Voraussetzung für eine sinnvolle *Frame*-Behandlung.

Auch die Benutzer-Schnittstelle wird durch *Frames* komplexer, da z. B. die Funktion `select` zum Bestätigen einer Selektion nicht über *Bookmarklets* aufgerufen werden kann, da diese nur auf die oberste *Frame*-Ebene zugreifen. Daher sollte *hier* ein Kontextmenü in JavaScript eingesetzt werden.

Anmerkung:
Das W3C hat in der Spezifikation von XHTML drei Dokument-Typen definiert. Der Typ

[1] Secure Socket Layer, ein Standard zur verschlüsselten Kommunikation

strict enthält überhaupt keine *Frames* mehr. Sie wurden von *Netscape* erfunden und wurden nur wegen ihrer massiven Nutzung in den HTML-Standard aufgenommen. Das W3C bemüht sich, *Frames* wieder zu vermeiden, da sie die Basismetapher „ein Dokument = eine Webseite“ verwässern.

JavaScript

Nicht alle zu extrahierenden Inhalte liegen in PH vor. Verweisziele werden z. B. oft per JavaScript erzeugt und stellen so ein Problem für Wrapper dar. *Kuhlins* und *Tredwell* fassen zusammen:

> Most of the current toolkits are unable to automatically overcome this type of hurdle and require user assistance in order to correctly navigate websites. Solving this problem would lead to major usability improvements for the toolkits. [KT03]

Damit ein Wrapper dynamisch erzeugten Text extrahieren kann, reicht es nicht aus, die JavaScript-Routinen einem Browser entsprechend ausführen zu können. Darüber hinaus muss der Benutzer in der Entwicklungsumgebung auch einen Weg haben, diesen Extraktionswunsch auszudrücken. Dazu muss automatisiert im JavaScript-Programmtext der Quellseite eingegriffen werden. Werkzeuge, die JavaScript ausführen können, nutzen meist die frei verfügbare *Rhino-Engine*. Das grundsätzliche Problem ist, dass die Interaktion von JavaScript mit dem Browser in Java nachgebildet werden muss. Es muss gegenüber der JavaScript-Programmier-Schnittstelle ein virtueller Browser dargestellt werden, welche mit einer Java-Programmier-Schnittstelle wiederum zugänglich ist. Es entsteht notwendigerweise ein neuer Typ Browser, der Unterschiede zu marktbeherrschenden Browsern aufweisen wird.

JavaScript
`onClick`-Handler respektieren

Bestehende `onClick`-Handler sollen erhalten bleiben. Dazu müssen bestehende JavaScript-Handler durch ein Kontextmenü ersetzt werden, in dem der Benutzer auswählt, ob er die ursprüngliche Funktion ausführen oder das Element selektieren möchte.

8.2 Erweitern der Funktionalität

8.2.1 Erkennung von Struktur und Inhalten

Bei Seiten, die, wie in Abschnitt 2.3 definiert, aus einer Datenbank mit Hilfe eines Schablonensystems erzeugt werden, kann für eine Testmenge von verschiedenen daraus erzeugten XH-Dateien ein *unifizierendes* XML-*Schema* gefunden werden. Dieses kann mit einer DTD oder einem XML-Schema ausgedrückt werden. Aufbauend auf dem größten unifizierenden XML-Schema kann der *größte gemeinsame Schnittbaum* der XH-Seiten gefunden werden. Elemente, die auf allen Seiten gemeinsam vorkommen, werden als statisch betrachtet. Der Schnittbaum enthält auch den Inhalt aller Elemente im unifizierenden XML-Schema, die gleich sind. Der größte Schnittbaum sollte sich – einen guten PH-Zerteiler vorausgesetzt –

genau mit den statischen Teilen der Seiten decken, also anwenderseitig das Schablonensystem wieder erkennen, welches die Seiten generiert hat. Das unifizierende XML-Schema kann zur Vereinfachung der Selektion (8.2.2) genutzt werden, der größte Schnittbaum hilft bei der Wrapper-Validierung (8.2.3).

8.2.2 Vereinfachungen der Selektion

URL-Darstellung in der Statuszeile Von relativen URLs wird nur der angegebene relative Teil in der Statuszeile angezeigt. Die meisten Browser zeigen an dieser Stelle absolute URLs.

Tabellen-Assistent Die Auswahl von ganzen Tabellenzeilen oder Spalten kann in der Benutzer-Schnittstelle vereinfacht werden. Dazu ist in JavaScript z. B. ein Kontextmenü zur Selektion der ganzen Zeile zu implementieren. Alternativ können auch zusätzliche kleine Zellen am Rand der Tabelle eingebaut werden, die beim Anklicken ihre Zeile bzw. Spalte selektieren. Der Entwurf des Moduls *ui* stellt frei, wie viel Logik anwenderseitig in JavaScript und wie viel serverseitig in Java implementiert wird.

Assistent für Datensatz-Seiten Bei Datensatz-orientierten Seiten (Definition in 2.4.2) ist der Benutzer evtl. nur an den dynamischen Inhalten aus der Datenbank interessiert. Hier könnte mit Hilfe der in 8.2.1 beschriebenen Erkennung direkt genau die dynamischen Inhalte vorselektiert werden, vorausgesetzt der Benutzer stellt dem System hinreichend viele – vermutlich reichen bereits sehr wenige – Testseiten zur Verfügung.

8.2.3 Wrapper-Validierung

Änderungen an der Quellseite sollen vom Wrapper möglichst erkannt werden, um nicht irrtümlich falsche Extraktionsergebnisse zu liefern. Dazu werden die erhaltenen HTTP-Antworten in XML-Form analysiert. Wenn auf der Quellseite relevante Änderungen stattfanden, ist ein neuer Wrapper zu erstellen. Mit relevanten Änderungen sind Änderungen an der Struktur des XH-Baumes gemeint, die dazu führen, dass die XPath-Pfade der Extraktion nicht mehr auf die richtigen Elemente zeigen.

Auch Änderungen an den statischen Elementen können relevant sein. Wenn beispielsweise auf einer Seite zur Hochwasserwarnung eines Tages der Hinweis „Diese Seite wird nicht mehr gepflegt, bitte benutzen sie http://...“ auftaucht, sollte der Wrapper aktualisiert werden.

Um diese Änderungen zuverlässig zu erkennen, muss für eine Seite definiert werden, welche Teile des HTML-DOMs sich ändern dürfen und welche nicht. Für Datensatz-orientierte Seiten kann hier der in 8.2.1 definierte größte gemeinsame Schnittbaum verwendet werden. Wenn ein Wrapper diesen speichert, kann er vor der Extraktion verifizieren, ob die Seite noch hinreichend ähnlich zu ihrer früheren Version ist. Bei einem Extraktionsvorgang wird die Seite dem Verifizierer

übergeben, welcher versucht, die neue Seite mit dem Schnittbaum zu überdecken. Sobald dies nicht gelingt, hat der Verifizierer eine strukturelle Änderung erkannt. Bei einem falschen Alarm, also irrelevanten Änderungen des statischen Teils, wird die neue Seite zur Menge der Musterseiten hinzugefügt und ein neuer, kleinerer – das geänderte Element wird nun auch als dynamisch betrachtet – Schnittbaum berechnet und gespeichert. Aus diesen Überlegungen heraus benötigen die Wrapper keinerlei Fehlertoleranz und Anpassungsfähigkeit gegenüber strukturellen Seitenänderungen.

8.2.4 Ergebnis-Analyse

Den Benutzer eines Wrappers interessiert, in wie weit sich die zugrunde liegende Seite oder das Extraktionsergebnis vom letzten Ergebnis unterscheidet. Mögliche Ergebnisse einer solchen Analyse sind:

- Modul *surf* konnte HTTP-Antwort nicht erhalten
- Modul *clean* konnte PH nicht zerteilen
- HTTP-Antwort identisch (seit x Aufrufen)
- HTTP-Antwort verändert, HTML-Seite identisch (seit x Aufrufen)
- HTML-Seite verändert, statische Struktur unverändert
- statische Struktur unverändert, Extraktionsergebnis verändert
- statische Struktur verändert (siehe auch 8.2.3)

Sollte sich die HTTP-Antwort oder die HTML-Seite seit einigen (je nach Quellseite) Aufrufen nicht verändert haben, so wird sie möglicherweise nicht mehr aktualisiert und der Benutzer ist zu warnen.

8.2.5 Automatischer Reparaturvorschlag

Da sich Webseiten ändern, ist die automatische Reparatur eines Wrappers wünschenswert [KLMM00]. Allerdings sollte nur ein automatisch erstellter Vorschlag gemacht werden und vom Benutzer bestätigt oder korrigiert werden. Treffend beschreibt [Tec] einen Ansatz:

> In addition, when the system creates an agent it also creates a set of rules to monitor the data being extracted. When an agent stops working appropriate notification can be sent and in some cases the agents can be fixed automatically.

Für eine automatische Reparatur muss eine der Musterseiten (siehe 8.2.3) komplett gespeichert werden. Eine als strukturell verändert erkannte Quellseite wird dann mit der alten verglichen.

In der Musterseite können die bisher vom XSLT-*Stylesheet* referenzierten Elemente identifiziert werden. Sie werden anschließend in der neuen Seite gesucht,

wobei nur Heuristiken helfen können. Zum Beispiel kann hier die minimale Editierdistanz auf den beiden XH-Bäumen verwendet werden. Zur Problematik des XML-Baumabstandes siehe auch Abschnitt 5.3.4.

Anschließend werden die XPath-Ausdrücke anhand der Informationen aus dem minimalen Editier-Skript angepasst. Der Benutzer muss lediglich gefragt werden, ob die angepassten XPath-Ausdrücke korrekt sind. Diese Abfrage kann visuell geschehen, wenn dem Benutzer eine Beispielseite gezeigt wird, auf der die von den neuen XPath-Ausdrücken referenzierten Elemente eingefärbt dargestellt sind.

Zum Vergleich sollte dem Benutzer die gespeicherte Quellseite gezeigt werden, auf der die selektierten Elemente ebenfalls hervorgehoben sind. Nur so weiß er, welches bei Wrapper-Erstellung die ausgewählten Elemente waren und kann visuell im Browser vergleichen und korrigieren.

Mehrschichtige Extraktion für komplexe Wrapper

Um Wrapper noch einfacher reparieren zu können, ist es möglich die Extraktion nicht in *einem* XSLT-*Stylesheet*, sondern in mehreren Schichten durchzuführen. Dabei wählt der Benutzer zunächst das Element, in dem die gewünschten Daten liegen grob aus und selektiert z. B. die Tabelle, in der die relevanten Daten liegen. Das System kann dann bereits eine Extraktion durchführen und präsentiert dem Benutzer im nächsten Schritt die Tabelle. Aus dieser selektiert er z. B. die relevante Zeile und führt einen weiteren Selektionsschritt durch. In einem letzten Schritt wählt er aus der Tabellenzeile den gewünschten Wert aus.

Der Vorteil dieses Verfahren liegt darin, dass das System die Ideen zur automatischen Reparatur und Validierung auch für jede einzelne Schicht durchführen kann, komplexere Wrapper so aber leichter zu reparieren sind. Hat der Seitenautor beispielsweise eine zweite Kopfzeile auf der Tabelle eingefügt, so kann das System erkennen, dass der Wrapper auf dieser Ebene repariert werden muss. Der Benutzer selektiert dazu lediglich in der Tabelle erneut die richtige – nun verschobene – Zeile und hat den Wrapper bereits vollständig repariert.

Die verschiedenen XSLT-*Stylesheets* können über Optimierungstechniken zu einem einzigen XSLT-*Stylesheet* zusammengefasst werden, um die Ausführungszeit zu verkürzen. Die einzelnen XSLT-*Stylesheets* werden nur zur Reparatur benötigt.

8.3 Höhere Dienste

Hier werden die Erweiterungen beschrieben, die auf der Extraktion von XML aus einzelnen PH-Seiten aufbauen.

Mehrseitige Extraktion

Eine nahe liegende Aufgabe ist das Aggregieren von Informationen mehrerer Webseiten, z. B. sind die Resultate von Suchanfragen oft auf mehrere Seiten verteilt. Der Benutzer muss dem System dazu in einem zweiten Schritt – nach der Wrapper-Erstellung für eine Seite – die Navigationsregeln „beibringen“. Er könnte z. B. mit einem *Bookmarklet* „`follow`“ den Proxy veranlassen, den nächsten Klick im HTML-Baum als Anweisung zu interpretieren. Der neue Wrapper

würde dann solange dem „`Next`“-Verweis einer Seite folgen, wie der Benutzer angibt (z. B. in einem *Popup-Window*) oder bis kein solcher mehr vorhanden ist. Ein Zusammenführen verschiedener Dokumente kann mit einem einzigen XSLT-*Stylesheet* ausgedrückt werden.

Automatische Analyse des Session Managements

Bei vielen Webangeboten muss der Benutzer sich zu Beginn mit einem Namen und Passwort authentifizieren, bevor er Daten suchen oder andere Aktionen tätigen kann. Danach navigiert der Benutzer zu einer Seite, von der er Daten extrahieren möchte.

Ein guter Wrapper sollte das Anmelde- und Navigationsverhalten simulieren können, um die gewünschten Daten extrahieren zu können. Das simple Aufzeichnen der direkt vorangehenden HTTP-Anfrage reicht nicht aus, da in dieser z. B. eindeutige Nummern kodiert sind, die der Benutzer einige HTTP-Anfragen zuvor auf seine Anmeldung hin erhalten hat. Solche Nummern zum *Session Management* haben üblicherweise nur einmalige Gültigkeit, ebenso *Cookies* die zur Authentifikation ebenfalls verwendet werden. Idealerweise kann ein Wrapper das Anmelde- und Navigationsverhalten des Benutzers „belauschen“ und automatisch nachahmen. Dazu ist eine gewisse „Transferleistung“ notwendig, denn manche der Daten wurden vom Benutzer eingetragen (über Formulare) und andere wurden vom Server generiert und in Verweise kodiert.

Die Idee der automatischen Analyse besteht darin, *Reverse Engeneering* der HTML-Seiten zu betreiben und so das *Session Management* zu entschlüsseln und nachahmen zu können. Zwei wichtige Tatsachen helfen:

Annahme 1: HTTP kennt keinen Zustand. Daher muss der Client den Zustand mit jeder Anfrage übermitteln. Der einzige Mechanismus, der von Browsern unterstützt wird, sind *Cookies*: Kleine vom Server definierbare Datenpakete, die der Browser mit jeder Anfrage – abhängig von der Anfrage-Adresse – wieder an den Server übermittelt.

Annahme 2: Jede andere Information kommt entweder vom Benutzer direkt (über Formulare) oder wurde vom Server in die letzte Seite generiert (versteckte Formularfelder und kodierte Verweise).

Das Modul *surf* verhält sich bei Sitzungs-*Cookies* bereits genau wie ein Browser. Damit können alle gesendeten Daten auf ihre Herkunft überprüft werden: Wenn sie sich nicht in der letzten Seite befinden, sind sie vom Benutzer, andernfalls vom Server. Daten vom Benutzer sind als statisch zu betrachten und sollen beim simulieren genau so wiedergegeben werden. Server-generierte Daten werden als dynamisch angesehen und aus der vorangehenden HTTP-Antwort übernommen. Schwierigkeiten bekommt dieser Ansatz evtl. bei mit JavaScript generierten oder modifizierten Daten. Wenn die HTTP-Anfragen als HTML aufbereitet werden, kann der Benutzer sogar über die bestehenden Extraktions-Schnittstellen angeben, welche Daten wie zugeordnet werden sollen, falls die Heuristik sich irren sollte.

9 Zusammenfassung, Bewertung und Ausblick

9.1 Zusammenfassung

Einleitung

Am Anfang der Arbeit wird die Notwendigkeit zur anwenderseitigen Datenextraktion von XML aus Webseiten erläutert und die Aufgabenstellung, die Erstellung eines Systems zur Erstellung und Ausführung von Wrappern (d_2c), vorgestellt.

Grundlagen

Im Kapitel „Grundlagen“ wird die Rolle eines Browser als Vorbild für Wrapper und ihre Erstellung motiviert. Zum einen ist ein Browser für den Benutzer die gewohnte Schnittstelle zur Nutzung von Webseiten; zum anderen stellen die marktbeherrschenden Browser den *de facto* Standard für die Interpretation von in der Praxis verwendetem HTML (PH) dar, welches sehr selten standardkonform ist.

80 % der Webseiten werden dynamisch durch Schablonensysteme (Struktur) mit Informationen aus Datenbanken (Inhalte) erzeugt. Wrapper nutzen die als *Struktur* definierten gemeinsamen Elemente einer Menge von Seiten aus, um die darin enthaltenen Inhalte zu extrahieren. Wrapper werden in der Literatur uneinheitlich definiert; die verwendete Definition beschreibt einen Wrapper als Software-Artefakt zur Extraktion von (Teil-)Dokumenten aus Webseiten. Die Abstraktion von den Basistechnologien HTTP und HTML durch Wrapper wird als *atomarer Dienst* bezeichnet, darauf aufbauende Schichten als *höhere Dienste*. Um von neuen Implementierungen profitieren zu können, werden *atomare Dienste* weiter in drei Schichten zerlegt: Laden der Quellseite, Extraktion und Ausgabe. Als Ausgabeformat bietet sich die speziell zum Austausch von Baumstrukturen konzipierte Sprache XML an. Das Ergebnis des Wrappers kann als XML-Dokument über eine Webschnittstelle bequem in anderen Systemen integriert werden.

Stand der Technik

Im Kapitel „Stand der Technik“ werden bei bestehenden Systemen als Schwäche vor allem die fehlende Aufteilung der Wrapper-Aufgabe in Schichten identifiziert. Viele Systeme extrahieren Daten direkt aus dem PH-Quelltext oder verwenden PH-Zerteiler zweifelhafter Güte. Keines der vorgestellten Systeme realisiert eine WYSIWYG-Schnittstelle im Browser. Von den vorgestellten Komponenten zur Zerteilung von PH werden *JTidy*, *CyberNeko* und *TagSoup* als Kandidaten zur Verwendung in d_2c identifiziert.

Entwurf

Im Kapitel „Entwurf“ werden die drei Aufgaben – WYSIWYG-Schnittstelle, Browser-ähnliche Zerteilung von PH und Ausgabe als XML – gelöst. PH wird in d_2c als wohlgeformtes XML (XH) dargestellt. So kann durch Extraktion mit

XSLT leicht XML ausgegeben werden. Ein extrahierendes XSLT-*Stylesheet* kann direkt aus einem annotierten XML-Dokument erzeugt werden. d_2c instrumentalisiert die Seite, wodurch im Browser durch anklicken die gewünschten Elemente ausgewählt werden können. Die einzelnen Schritte der Wrapper-Erstellung sind über *Bookmarklets* ebenfalls im Browser steuerbar.

Es wird eine adäquate Architektur in zwei Teilssystemen entwickelt. Das Produktionssystem führt Wrapper aus. Dazu nutzt es seine Module: zur Abstraktion von HTTP (*surf*), Abstraktion von HTML (*clean*) und XML-Transformation (*extract*). Erstellt werden die Wrapper mit einem anderen Teilsystem, dem Entwicklungssystem. Es besteht aus einem filternden Proxy (*proxy*), der an ein Modul zur Abstraktion der Benutzer-Schnittstelle (*ui*) und eines zur XSLT-*Stylesheet*-Erzeugung (*learn*) delegiert.

Wiederverwendung

Im Kapitel „Wiederverwendung“ wird ein Benchmark entwickelt, der PH-Zerteiler untereinander vergleicht. Dazu wird ein Verfahren entwickelt, um den Zerteilungsbaum eines Browser zu erhalten. Mit zwei verschiedenen XML-Ähnlichkeitsmaßen wird *TagSoup* als derjenige Zerteiler ermittelt, der die Zerteilung am ähnlichsten zum verbreiteten Browser *Opera* durchführt und daher im Modul *clean* verwendet wird. Ebenfalls in diesem Kapitel ausgewählt, werden ein HTTP-Client, ein Webserver (als Teil von *proxy*) und ein DOM-Paket.

Umsetzung

Im Kapitel „Umsetzung“ werden die in der Architektur definierten Module durch Klassendiagramme verfeinert. Den größten Aufwand bei der Implementierung verursachen die Datenstrukturen zur Kommunikation zwischen den Modulen und zur Abstraktion von den HTTP- und HTML-Schnittstellen wiederverwendeter Komponenten.

Evaluation

Das Kapitel „Evaluation“ beschreibt im Detail die Erstellung eines Wrappers und Testmöglichkeiten aus Sicht des Benutzers. Ein Vergleich mit den Anfangs definierten Kriterien zeigt, dass d_2c den Anforderungen an eine WYSIWYG-Schnittstelle im Browser gerecht wird. Durch den Benchmark kann eine möglichst Browser-ähnliche Interpretation von PH erreicht werden. Als Anwendungsmöglichkeiten des Systems werden primär Informations-integrierende Dienste wie Meta-Suchmaschinen oder Preisvergleichsdienste gesehen, bei denen das Kernproblem in der Erstellung und Verwaltung von hunderten von Wrappern liegt.

Vorgesehen Erweiterungen

Im Kapitel „Vorgesehene Erweiterungen“ werden Vorschläge für eine höhere Abdeckung von Webseiten und Webservern gemacht. Dazu sollen beispielsweise *Frames*, *SSL* und möglichst auch *JavaScript* beherrscht werden. Zur Erweiterung der Funktionalität von *atomaren Diensten* zählen Verbesserungen der Benutzer-Schnittstelle und eine Möglichkeit zur Analyse von Wrapper-Ergebnissen. Diese beruht auf einem Vergleich der früheren mit der aktuellen XML-Struktur, ähnlich dem beim Benchmark definierten *größten gemeinsamen Schnittschaum* von zwei XML-Dokumenten. Darauf aufbauend sind eine Wrapper-Validierung und ein systemgenerierter Reparaturvorschlag geplant. Geplante *höhere Dienste* sollen mehrseitige Extraktion und weitere Automatisierungsmöglichkeiten bieten.

9.2 Bewertung

Real vorkommendes HTML analog zu einem Browser aufbereiten zu können, um es automatisch weiter verarbeiten zu können, ist eine nützliche Technik zur Einbindung von Systemen und Datenbanken, die über eine HTML-Schnittstelle verfügen.

Das Konzept, Wrapper schnell erzeugen zu können und dafür auf die Robustheit zu verzichten, ist neu. Ein Wrapper, der möglichst selten falsche Daten liefern darf, kann keinen anderen Ansatz verfolgen. Allerdings muss er erkennen, wenn sich eine Quellseite relevant verändert hat (siehe 8.2.3). Dazu sind die in der Arbeit definierten Begriffe *Struktur* und *Inhalt* von Webseiten hilfreich.

d_2c ist durch seine einfach Bedienung von Laien verwendbar. Prinzipiell ist ein einfacherer Prozess weniger fehleranfällig und somit auch für den Experten ein Gewinn.

Die Unterscheidung von atomaren und höheren Diensten ist hilfreich für das Verständnis und die Entwicklung von Wrappern. Insbesondere lassen sich dadurch verschiedene Wrapper-Ansätze leichter miteinander kombinieren.

d_2c wandelt PH in XML und ist für andere Datenformate (z. B. unstrukturierten Text) daher nicht geeignet. Es verwendet – im Gegensatz zu allen 32 in [TK] genannten Werkzeugen – XSLT als Extraktionssprache.

Die Wahl von XSLT als Extraktionssprache ist kritisch zu sehen. Zwar kann ein XSLT-*Stylesheet* einfach erstellt werden, aber erst mit XSLT 2.0 werden auch reguläre Ausdrücke zur Selektion von Textteilen ermöglicht. Daher beschränkt sich die Extraktion auf komplette Textelemente. Positiv ist allerdings die Möglichkeit, bestehende Werkzeuge und Optimierungen des Gebietes XSLT/XML nutzen zu können. Dies ist ein Vorteil, den andere Wrapper-Systeme nicht haben.

Die modulare Architektur hat sich bewährt, da während der Entwicklungszeit drei verschiedene Implementierungen des Moduls *surf* und ebenso drei für das Modul *clean* testweise genutzt werden konnten.

Der entwickelte Benchmark zur Auswahl eines fehlertoleranten PH-Zerteilers – inklusive seines neuen XML-Ähnlichkeitsmaßes und der Möglichkeit, die Zerteilung eines Browser nutzen zu können – ist als zusätzliche Leistung unabhängig von d_2c nutzbar, um zu Messen, wie gut ein PH-Zerteiler einen Browser approximiert.

Der WYSIWYG-Ansatz im Browser funktioniert sehr gut und sollte weiter verfolgt werden. Auch eine Jobverwaltung und Testmöglichkeiten, mit denen die gespeicherten HTTP-Anfragen überprüft und die erzeugten XSLT-Transformationen getestet werden können, konnten leicht im Browser realisiert werden.

Die Benutzerführung über *Bookmarklets* ist für einen Einsatz als Portal-Dienst noch zu umständlich. Hier sollte z. B. eine Kopfzeile mit gleichen Optionen in die PH-Seiten eingefügt werden. Das System macht Änderungen an der Schnittstelle leicht möglich.

Die Umsetzung wurde ausreichend dokumentiert, um eine Weiterentwicklung des Systems zu ermöglichen.

9.3 Ausblick

Zukünftig werden immer mehr Systeme Schnittstellen besitzen, die von der direkten Benutzung des Internets abstrahieren. Bekannte Beispiele dafür sind:

- Meta-Suchmaschinen, die zahlreiche Suchmaschinen anfragen und die Ergebnisse zusammenfassen,
- Suchmaschinen-Eintrags-Helfer, die eine Seite automatisch bei einigen hundert Suchmaschinen anmelden,
- sogenannte *Sniper*-Programme, die automatisch in letzter Sekunde bei einer Online-Auktion ein Gebot abgeben,
- Preisvergleiche, die mittlerweile für nahezu jede Produktgruppe existieren. Sie extrahieren die Preis-Informationen aus den Webseiten von Online-Händlern.

Über das Extrahieren von Informationen hinaus wird in Zukunft eine komplette Abstraktion der WWW-Schnittstelle, z. B. zur Nutzung über Funktionsaufrufe in einem Programm, an Bedeutung gewinnen. d_2c geht einen Schritt in diese Richtung, indem es dynamische Anfrageparameter zur Laufzeit nutzt und eine entsprechend veränderte Anfrage an den Webserver schickt.

Wünschenswert wäre hier ein System, welches bei gegebener Webseite „ausprobiert“, welche Aktionen zur Verfügung stehen und was sie bewirken, um so völlig autonom einen Wrapper zu erzeugen.

Für alle Abstraktionen unbedingt notwendig ist ein zu verbreiteten Browsern möglichst ähnlicher PH-Zerteiler, idealerweise vergleichbar mit dem Zerteiler des wichtigen *Internet Explorers*. Da dieses Forschungsgebiet gerade in jüngster Zeit mit neuen Ideen und Implementierungen (z. B. *CyberNeko* und *TagSoup*) belebt wird, sind Fortschritte zu erwarten. Der in dieser Arbeit entwickelte Benchmark bietet ein messbares Kriterium.

A Installation

- Konfigurations-Datei (eine Java-`Properties`-Datei) anpassen
  Die Standardwerte sind angegeben.

```
##########################
# Configfile for d2c
##########################
# default values
########################
# rootPath = ..
# timeoutSeconds = 30
# followRedirects = true
# proxyPort = 7777
# webservicePort = 8888
# referrer = http://www.google.com
# userAgentName = Mozilla/4.0
  (compatible; MSIE 6.0; Windows NT 5.1; (R1 1.3))
# compare = false
```

- d_2c-Server in Java starten

```
    Java -classpath
 "dest:nekohtml.jar:lib/clean/nekohtmlXni.jar:lib/dom4j-fu
ll.jar:lib/log4j-1.2.7.jar:lib/commons-logging.jar:lib/sur
f/jce.jar:lib/surf/jsse.jar" ds.DevelopmentSystem
```

  Alles in einer Zeile, `classpath`-Argument ohne Leerzeichen, `JAR`-Dateien in einem Unterverzeichnis `lib`.

- Entwicklungssystem als Proxy im Browser eintragen

  Standardmäßig ist Port 8888 eingestellt.

- *Bookmarklets* installieren
  Dies ist je nach Browser etwas unterschiedlich, es wird im Prinzip einfach ein Lesezeichen angelegt, mit dem Namen des *Bookmarklets* und den hier angegebenen Verweiszielen.

Verwendete *Bookmarklets*:

rec
`javascript:open('http://record.action.d2c/?'+Math.random()`
`,'record','width=600,height=200,left=50,top=0');void 0`

select
`javascript:d2c_callServer();`

stopui
`http://stopui.action.d2c/`

save
`javascript:open('http://job.action.d2c/foo?cmd=save&foo='`
`+Math.random(),'record','width=600,height=400,left=50,`
`top=0');void 0`

run
`http://job.action.d2c/foo?cmd=run`

Sowie zum Testen:

surf
`http://job.action.d2c/foo?cmd=surf`

extract
`http://job.action.d2c/foo?cmd=extract`

- Testen durch Aufruf z.B: URL:
`http://`*Server*`:7777/d2c/foo?d2c_jobname=`*Auftragsname*
Optional zusätzlich: `&`*Parameter*`=`*Wert*

B Beispiele für Aufbereitung von PH zu XH

PH-Quelldatei (www.nvca.org)

```
<html>

    <head>
        <meta http-equiv="content-type"
          content="text/html;charset=iso-8859-1">
        <meta name="generator" content="Adobe GoLive 4">
        <title>Welcome to NVCA</title>
    </head>

    <frameset rows="90,*" framespacing="0" border="0">
        <frame src="top_frame.html" name="top" scrolling="NO" noresize>
        <frame src="home-frame2.html" name="content" noresize>
    </frameset>
    <noframes>

        <body>
        </body>

    </noframes>
</html>
```

HTML 4.01 Frameset-Definition

Aus der XHTML-Spezifikation [XHT02]:

> This is the HTML 4.01 Frameset DTD, which should be used for documents with frames. This DTD is identical to the HTML 4.01 Transitional DTD except for the content model of the „HTML“ element: in frameset documents, the „FRAMESET“ element replaces the „BODY“ element.

```
<html>
    <head>
    ...
    </head>
    <frameset>
    ...
    </frameset>
</html>
```

Von *Opera* erzeugtes XH

```
<html>
    <head>
        <meta http-equiv="content-type"
          content="text/html;charset=iso-8859-1"/>
        <meta name="generator" content="adobe golive 4"/>
        <title>welcome to nvca</title>
    </head>
    <body>
        <frameset rows="90,*" framespacing="0" border="">
            <frame src="http://192.168.0.21/top_frame.html"
              name="top" scrolling="no" noresize=""/>
            <frame src="http://192.168.0.21/home-frame2.html"
              name="content" noresize=""/>
        </frameset>
        <noframes>




    </noframes>
        <script id="d2c_inserted" type="text/javascript"/>
    </body>
</html>
```

Von *Internet Explorer* erzeugtes Xml

```
<html>
    <head>
        <title/>
        <script id="d2c_inserted" type="text/javascript"/>
    </head>
</html>
```

Das Script wurde hier an anderer Stelle eingefügt – der Internet Explorer führt es sonst nicht aus.

Von *CyberNeko* erzeugtes XML

```
<html>
    <head>
        <meta content="text/html;charset=iso-8859-1"
          http-equiv="content-type"/>
        <meta content="adobe golive 4" name="generator"/>
        <title>welcome to nvca</title>
    </head>
    <frameset border="0" framespacing="0" rows="90,*">
        <frame name="top" noresize="" scrolling="no"
          src="top_frame.html"/>
        <frame name="content" noresize=""
```

```
          src="home-frame2.html"/>
    </frameset>
    <frameset>
        <noframes>
            <body>
        </body>
        </noframes>
    </frameset>
</html>
```

Von *TagSoup* erzeugtes XML

```
<html xmlns="http://www.w3.org/1999/xhtml"
  version="-//w3c//dtd html 4.01 transitional//en">
    <head>
        <meta content="text/html;charset=iso-8859-1"
          httpequiv="content-type"/>
        <meta content="adobe golive 4" name="generator"/>
        <title>welcome to nvca</title>
    </head>
    <frameset border="0" framespacing="0" rows="90,*"/>
    <noframes>
        <body>   </body>
    </noframes>
</html>
```

Von *JTidy* erzeugtes XML

```
<html>
    <head>
        <meta content="html tidy,
          see www.w3.org" name="generator"/>
        <meta content="text/html;charset=iso-8859-1"
          http-equiv="content-type"/>
        <meta content="adobe golive 4" name="generator"/>
        <title>welcome to nvca</title>
    </head>
    <frameset border="0" framespacing="0" rows="90,*">
        <frame noresize="noresize" scrolling="no" name="top"
          src="top_frame.html"/>
        <frame noresize="noresize" name="content"
          src="home-frame2.html"/>
        <noframes>
            <body/>
        </noframes>
    </frameset>
</html>
```

C Benutzte Werkzeuge

Ethereal als *Port-Sniffer* für die Analyse der ein- und ausgehenden HTTP-Nachrichten, wie sie *wirklich* über die Netzwerkkarte gesendet werden.

TCPMonitor aus dem Apache *Axis Toolkit*, hilft ebenfalls beim Überwachen von HTTP-Verkehr.

Tiny Java Web Server als Test-Webserver, um das Gesamtsystem mit definiertem Serververhalten und definiertem Seiteninhalt testen zu können.

WinEdt als LaTeX-Editor für die Ausarbeitung

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[BFG01a] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. The elog web extraction language. In *Logic for Programming and Automated Reasoning (LPAR)*, pages 548–560, 2001.

[BFG01b] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *The VLDB Journal*, pages 119–128, 2001. Online: `citeseer.nj.nec.com/baumgartner01visual.html`.

[Coh99] William W. Cohen. Recognizing structure in web pages using similarity queries. In *AAAI/IAAI*, pages 59–66, 1999. Online: `citeseer.nj.nec.com/cohen99recognizing.html`.

[del01] Eero Lempinen and Harri Saarikoski (Republica Corp.), editors. DEL - data extraction language [online]. 10 2001. Online: `www.w3.org/TR/data-extraction`.

[FGC+97] Armando Fox, Steven D. Gribble, Yatin Chawathe, Anthony S. Polito, Andrew Huang, Benjamin Ling, and Eric A. Brewer. Orthogonal extensions to the www user interface using client-side technologies. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 83–84. ACM Press, 1997.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1994.

[GLdSRN00] Paulo Braz Golgher, Alberto H. F. Laender, Altigran Soares da Silva, and Berthier A. Ribeiro-Neto. An example-based environment for wrapper generation. In *ER Workshops*, pages 152–164, 2000. Online: `citeseer.nj.nec.com/golgher00examplebased.html`.

[htt] HttpUnit FAQ [online, gesichtet 22.04.2003]. Online: `httpunit.sourceforge.net/doc/faq.html`.

[Ite02] ItemField. Contentmaster architecture. White Paper, 2002. Online: `www.itemfield.com`.

[KLMM00] Craig A. Knoblock, Kristina Lerman, Steven Minton, and Ion Muslea. Accurately and reliably extracting data from the web: A machine learning approach. *IEEE Data Engineering Bulletin*, 23(4):33–41, 2000. Online: `citeseer.nj.nec.com/knoblock00accurately.html`.

[KT03] Stefan Kuhlins and Ross Tredwell. Toolkits for generating wrappers – a survey of software toolkits for automated data extraction from web sites. In Mehmet Aksit, Mira Mezini, and Rainer Unland, editors, *Objects, Components, Architectures, Services, and Applications for a Networked World*, volume 2591 of *Lecture Notes in Computer Science (LNCS)*, pages 184–198, 2003.

[Kus99] Nicholas Kushmerick. Regression testing for wrapper maintenance. In *AAAI/IAAI*, pages 74–79, 1999. Online: `citeseer.nj.nec.com/kushmerick99regression.html`.

[Kus00] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000. Online: `citeseer.nj.nec.com/kushmerick00wrapper.html`.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, February 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[Lin01] Tancred Lindholm. A 3-way merging algorithm for synchronizing ordered trees - the 3dm-merging and differencing tool for xml, sep 2001. Online: `citeseer.nj.nec.com/lindholm01way.html`.

[LL01] Mengchi Liu and Tok Wang Ling. A rule-based query language for HTML. In *Database Systems for Advanced Applications*, pages 6–13, 2001. Online: `citeseer.nj.nec.com/liu01rulebased.html`.

[LPH00] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *ICDE*, pages 611–621, 2000. Online: `citeseer.nj.nec.com/liu00xwrap.html`.

[LRNdSS00] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran Soares da Silva, and Elaine E. Silva. Representing web data as complex objects. In *EC-Web*, pages 216–228, 2000. Online: `citeseer.nj.nec.com/laender00representing.html`.

[LRNST02] A. Laender, B. Ribeiro-Neto, A. Silva, and J. Teixeira. A brief survey of web data extraction tools. In *SIGMOD Record*, volume 31-2, June 2002. Online: `citeseer.nj.nec.com/laender02brief.html`.

[MM02] R. Miller and A. Myers. Multiple selections in smart text editing, 2002. Online: citeseer.nj.nec.com/miller02multiple.html.

[Mou02] Adrian Mouat. *XML Diff and Patch Utilities*. PhD thesis, Heriot-Watt University, Edinburgh, Scotland, jun 2002. Online: sourceforge.net/projects/diffxml.

[Myl01] Jussi Myllymaki. Effective web data extraction with standard XML technologies. In *World Wide Web*, pages 689–696, 2001. Online: citeseer.nj.nec.com/452335.html.

[Nie02] Nielsen//NetRatings. Global internet trends Q4 2002, 2002.

[NJ02] Andrew Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)*, Madison, Wisconsin, USA, June 2002. Online: citeseer.nj.nec.com/nierman02evaluating.html.

[SA99] Arnaud Sahuguet and Fabien Azavant. Web ecology: Recycling HTML pages as XML documents using w4f. In *WebDB (Informal Proceedings)*, pages 31–36, 1999. Online: citeseer.nj.nec.com/sahuguet99web.html.

[SA01] Arnaud Sahuguet and Fabien Azavant. Building intelligent web applications using lightweight wrappers. *Data Knowledge Engineering*, 36(3):283–316, 2001. Online: citeseer.nj.nec.com/sahuguet00building.html.

[Sch] Ben Schwan. Allein gegen Netscape und Microsoft: Interview mit Opera-Chef jon s. von tetzchner. Online: www.heise.de/ct/01/10/047/default.shtml.

[SN03] Steffen Schott and Markus L. Noga. Lazy evaluation of XSLT. In *IVME 2003*. ACM Press, 2003. submitted.

[sww01] Final report: International semantic web working symposium (SWWS). jul 2001.

[Tec] Fetch Technoliogies. Technology overview: Reliably extracting web data. White Paper. Online: www.fetch.com.

[TK] Ross Tredwell and Stefan Kuhlins. Wrapper development tools [online, gesichtet 21.02.2003]. Online: www.wifo.uni-mannheim.de/ kuhlins/wrappertools/.

[XHT02] Xhtml 1.0 the extensible hypertext markup language (second edition) [online]. aug 2002. Online: www.w3.org/TR/xhtml1/.